
BYTE BOOKS /McGraw-Hill
Introducción al
MICROPROCESADOR
8086/8088 (16 Bit)
Christopher L. Morgan – Mitchell Waite

Introducción al

MICROPROCESADOR 8086/8088 (16 Bit)

Introducción al
MICROPROCESADOR 8086/8088 (16 Bit)

Christopher L. Morgan
Mitchell Waite

Traducción:

Jordi Aguilo Llobet
Director del Departamento de Informática
Facultad de Ciencias
Universidad Autónoma de Barcelona

Elena Valderrama Valles
Profesora agregada de Sistemas Lognos
Departamento de Informática
Facultad de Ciencias
Universidad Autónoma de Barcelona

Revisión técnica:

Antonio Vaquero Sánchez
Catedrático de Informática
Facultad de Ciencias Físicas
Universidad Complutense

BYTE BOOKS/McGraw-Hill

MADRID • BOGOTA • BUENOS AIRES • GUATEMALA • LISBOA • MEXICO • NUEVA YORK
PANAMA • SAN JUAN • SANTIAGO • SAO PAULO
AUCKLAND • HAMBURGO • JOHANNESBURGO • LONDRES • MONTREAL • NUEVA DELHI
PARIS • SAN FRANCISCO • SINGAPUR • ST. LOUIS • SIDNEY • TOKIO • TORONTO

INTRODUCCION AL MICROPROCESADOR 8086/8088 (16 Bit)

Atlacomulco 499-501, Naucalpan de Juárez, Edo. de México
Miembro de la Cámara Nacional de la Industria Editorial, Reg. núm. 465.

ISBN: 968-451-628-2

Traducido de la primera edición en inglés de

8086/8088 16-BIT MICROPROCESSOR PRIMER

ISBN: 0-07-043109-4

Edición exclusiva para ediciones La Colina, S. A. (España)

ISBN: 84-85240-97-9
Depósito legal: M. 25.597-1984

Compuesto en Fernández Ciudad, S. L.
Impreso en Gráficas EMA. Miguel Yuste, 27. MADRID

PRINTED IN SPAIN - IMPRESO EN ESPAÑA

Agradecimientos

Cualquier libro digno de ser leído necesita contribuciones mucho más allá de las de los propios autores. Esto se hace más obvio cuando los autores han acabado su primer bosquejo. Tal vez porque han creado, amado y «educado» el tema durante tanto tiempo, les es muchas veces imposible evaluar lo que han escrito. Sabiendo que llegaríamos a esta etapa dada la inmensidad del tema, los autores echan mano de amigos, esposas, estudiantes, consultores y profesionales informáticos para que lo evalúen, critiquen, sugieran mejoras y corrijan errores. Estaremos siempre en deuda con todos ellos y nos gustaría agradecérselo desde aquí.

Primero y más importante, nos gustaría agradecernos mutuamente los esfuerzos. Nuestro respeto mutuo y admiración por la destreza y contribuciones del otro fue el ingrediente principal, no sólo para hacer de este libro un gran libro, sino también un libro inspirado.

Por su extremada paciencia y estímulo en las muchas revisiones del manuscrito, y por soportar las numerosas llamadas, cuestiones y mensajes entre los autores, nos gustaría dejar constancia de nuestra deuda de gratitud con Carol Morgan. Estamos particularmente en deuda con ella por la tremenda cantidad de correcciones de pruebas y valiosas sugerencias de cómo hacer el libro más legible, ¡todo ello mientras criaba a un chiquillo y nacía otro!

A Alan Orcutt le damos nuestras más sinceras gracias por todas las aportaciones. Como estudiante del Dr. Morgan escribió comentarios para los programas-ejemplo de los Chips Controladores Programables de Interfase 8255 y 8251. Tras su graduación, siendo ingeniero comercial de Intel Corporation, proporcionó la ligazón necesaria con Intel y la oportunidad de utilizar software crucial.

Alan pasó muchas horas ayudándonos a preparar los programas ejemplo para el NDP 8087, trabajando con un Sistema de Desarrollo de Intel, y obteniendo los programas ensamblados del Programador de E/S 8089.

Por sus valiosas revisiones y comentarios del primer manuscrito, queremos darle las gracias a Marvin Winzenread, Alan Orcutt, Philip Lieberman y David Fox. Estos expertos convirtieron un manuscrito interesante y ligeramente confuso en un gran manuscrito, equilibrado y claro. Nos gustaría también pagar nuestra deuda de gratitud con David Fox por el tiempo extra que invirtió en ejecutar los ejemplos del 8088 del capítulo 7 sobre su computador Compupro. Sus comentarios fueron una ayuda inmensa

para convertir estos programas en las mejores herramientas didácticas.

Nos gustaría también agradecer a Bruce Roberts y Tom McMillan de Byte Publications el cuidado y rápido trabajo de edición del manuscrito.

Gracias a Intel por sus estímulos y cooperación en la escritura del libro. Sus chips están realmente entre los procesadores de 16 bits más potentes, populares y mejor soportados del mercado. Esto hizo que nuestro libro fuese interesante de escribir, y su buena documentación nos permitió «escarbar» en el tema tan profundamente como queríamos mientras lo investigábamos. Su técnica de escritura se coloca por encima de muchas otras, como un ejemplo de «como debe hacerse». ¡Les deseamos un éxito tremendo con sus fantásticos chips!

Christopher L. Morgan y Mitchell Waite.

Contenido

Prólogo

Los nuevos y potentes microprocesadores de 16 bits de Intel —el 8086 y el 8088— representan la última innovación en circuitos integrados de estado sólido. Se espera que estos nuevos microprocesadores eclipsen a los populares microprocesadores de 8 bits del pasado, convirtiéndose en la elección más acertada para todos aquellos productos basados en micros. La velocidad de proceso del 8086 de 16 bits puede ser, según los casos, de 2 a 4.000 veces mayor que la de su predecesor de 8 bits. La ganancia «moderada» de velocidad ocurre cuando se realizan instrucciones aritméticas de 16 bits con enteros, y el valor máximo puede darse fácilmente en cálculos de doble precisión y coma flotante si se asocian en paralelo la CPU 8086 y el procesador numérico 8087. Además, la cantidad de código en programas equivalentes puede llegar a ser un 50 por 100 menor en los microprocesadores de 16 bits. De hecho, los nuevos microprocesadores llevan incorporadas rutinas de multiplicación y división de enteros, instrucciones de tratamiento de cadenas y otras características que convierten al juego de instrucciones del microprocesador en casi un lenguaje de alto nivel.

Los Procesadores de Propósito General 8086 y 8088 utilizan conceptos de arquitectura avanzados tales como la gestión de memoria, interrupciones vectorizadas multinivel y procesamiento paralelo que explicaremos a lo largo de este libro. Estas características proporcionan a los nuevos chips de 16 bits una gran ventaja sobre los anteriores microprocesadores de 8 bits, permitiendo su uso en combinaciones sinergéticas en las cuales el procesador final es más potente que la suma de sus componentes individuales.

Además de cubrir los conceptos básicos de los microprocesadores de 16 bits, el libro describe cómo ciertos procesadores pueden trabajar en paralelo con el 8086/8088, incrementando varias veces su rendimiento. En particular, se estudia el «increíble» Procesador de Datos Numérico 8087 (NDP 8087), y el superflexible Procesador de Entrada/Salida 8089 (IOP 8089). Funcionando en paralelo con el 8086, estos chips aumentan el rendimiento del 8086 al realizar por él ciertas tareas que requerirían un alto consumo de tiempo. El 8087 puede ejecutar potentes instrucciones matemáticas por hardware, mucho más rápidamente que si se hicieran por software en el 8086. El NDP 8087 aumenta el juego de instrucciones de la CPU 8086, ofreciendo al programador el equivalente a tener incorporada una calculadora científica equipada con funciones trigonométricas, logarítmicas y

otras de carácter básico. Además, el 8087 ¡puede representar y trabajar con números de hasta 10^{4000}! El 8089 es capaz de realizar inteligentes operaciones de E/S intercaladas en canal doble al mismo tiempo que la CPU continúa con el programa principal. Con la adición de estos dos procesadores en paralelo, el 8086, o el 8088, pueden ver aumentada 100 veces su potencia de cálculo.

Este libro es el primero que explica, de una forma clara y eminentemente práctica, cómo funcionan los increíbles microprocesadores de 16 bits 8086 y 8088 y sus procesadores paralelos. Explica asimismo el Intel 186 (se trata de una versión mejorada y actualizada del 8086), el Intel 286 (un 8086 con gestión de memoria incorporada) y el prometedor Intel 432 (el superpoderoso procesador de 32 bits).

El primer capítulo comienza explicando qué es un microprocesador de 16 bits y da una visión general del contenido de este libro. Detalla qué es lo que hace tan especiales a los microprocesadores de 16 bits, explora el mercado de posibles productos con microprocesadores de 16 bits y da una primera idea del 8086 y del 8088. Incluye también una revisión del software existente y una descripción de la organización del libro.

A continuación se introducen los conceptos básicos de los microprocesadores de 16 bits, mecanismos tales como la incorporación de periféricos gráficos a un microcomputador típico de 16 bits, los tipos de datos y números, la organización física de la memoria, la gestión de memoria, multiproceso, procesamiento en paralelo y las diferencias esenciales entre el 8086 y el 8088. Se describen también los diferentes niveles de programación, desde los lenguajes de alto nivel hasta los ensambladores, el microcódigo y el nanocódigo, y se desarrollan algunos programas en lenguaje ensamblador para microprocesadores de 16 bits.

En el capítulo 3, el libro introduce los Procesadores de Propósito General 8086 y 8088, estudiándolos en profundidad y describiendo sus características eléctricas, estructura, juego de instrucciones, etc. Los capítulos siguientes tratan el Procesador de Datos Numérico 8087 y el Procesador de Entrada/Salida 8089. Se incluye también un capítulo sobre la familia completa de chips de soporte del 8086, para que se pueda optimizar el uso de estos chips desde una única fuente. El capítulo 7 versa sobre la programación del 8086. Presenta 11 programas-ejemplo, cortos y didácticos, para que el lector pueda codificarlos y probarlos. Estos programas se han simplificado y optimizado, y cubren temas como el colocar y mover letras por la pantalla, contar sobre una pantalla, un test rápido de memoria, volcado de los registros EU, impresión de una sola línea, etc. Finalmente, hay un último capítulo dedicado al hardware y software disponible hoy por hoy para estos microprocesadores, y a la nueva computadora personal de IBM. Los apéndices cubren los nuevos chips de Intel 432 (el sofisticado chip de 32 bits), 186 y 286.

A lo largo del libro se han incluido numerosos diagramas y programas-ejemplo como ilustración a las operaciones. Los esquemas tri-dimensionales añaden claridad a las explicaciones.

Se han omitido los diagramas de tiempo y toda aquella información que puede obtenerse en los manuales de referencia de los fabricantes o en libros más avanzados. Hemos intentado una aproximación visual, presentando el tema de los microprocesadores de 16 bits de una forma interesante, legible y fácilmente asimilable. Todos los programas del libro (excepto los últimos) son cortos y concisos. Se desarrolló una secuencia completa en una placa procesadora Dual 8085/8088 para el bus S-100, con el sistema operativo CP/M y el ensamblador cruzado de Microsoft. La depuración de los programas se hizo en Intel.

El 8086 y 8088 son los primeros individuos de una nueva raza, la de los super-potentes chips microprocesadores de 16 bits de la tercera generación. Comparativamente hablando, son algo así como los Ford Modelo T de las computadoras. Puede estar seguro que los productos nacidos de estos nuevos chips harán dar un salto de gigante a las capacidades de las computadoras de bajo costo. Confiamos sinceramente que este libro le ayude a comprender la «aterradora» potencia de estos dispositivos de 16 bits y de los productos derivados.

Christopher L. Morgan, Ph. D.
Mitchell Waite

Capítulo 1

Perspectivas

¿QUE ES UN MICROPROCESADOR DE 16 BITS?

Los procesadores de 16 bits son una nueva generación de chips microprocesadores desarrollados por los «magos del silicio», los fabricantes de circuitos integrados. Son ingenios más grandes y poderosos, destinados a reemplazar o completar a las microcomputadoras de 8 bits de los años setenta, que fueron las que comenzaron la revolución de las microcomputadoras. Estas nuevas microcomputadoras de 16 bits son muy interesantes, entre otras cosas porque, para ciertas configuraciones, y en ciertos cálculos, pueden llegar a ser hasta 4.000 veces más potentes que los chips de 8 bits, a un coste relativamente igual. Este asombroso poder de cálculo va a desembocar en el desarrollo y comercialización de nuevas máquinas más inteligentes que causarán un gran impacto en la sociedad. Estos nuevos microprocesadores de 16 bits permitirán aumentar significativamente la «inteligencia» de todo dispositivo procesador. Así, es de esperar una nueva revolución en los «productos finales» que los fabricantes y diseñadores lanzarán al mercado en los próximos años: no será de extrañar que pronto aparezcan máquinas con posibilidad de hablar y

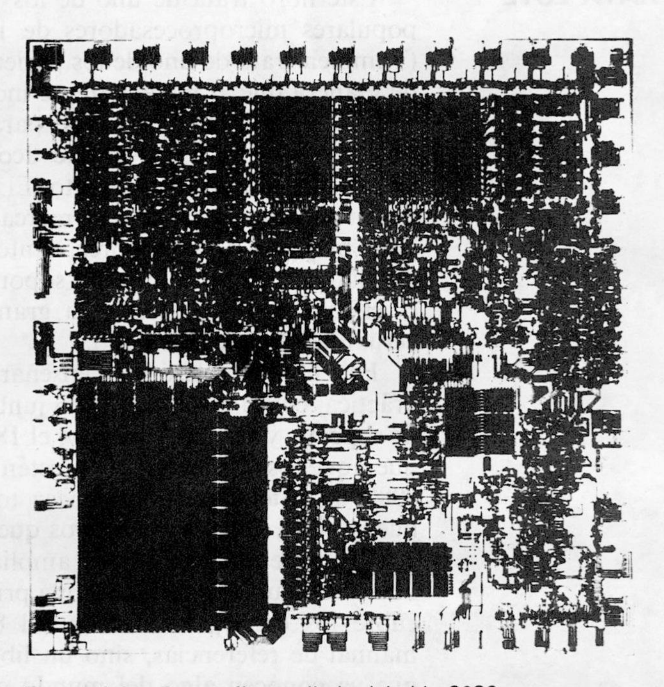


Figura 1.1a: Fotografía ampliada del chip 8086.

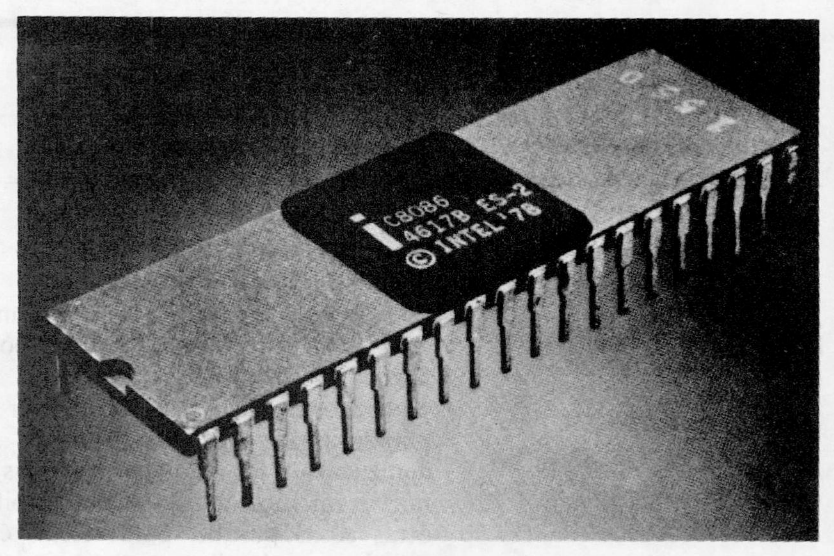


Figura 1.1b: El 8086 una vez encapsulado.

escuchar, pantallas de color tridimensionales, redes de comunicación avanzada que permitan acceder a bases de datos gigantescas a través de la red telefónica, automóviles computarizados y astutos controladores caseros que sigan el movimiento de los habitantes de la casa para ajustar al mínimo el consumo de energía.

¿DE QUE TRATA ESTE LIBRO?

Este libro trata de uno de los primeros, más potentes y más populares microprocesadores de 16 bits, el chip 8086 de Intel. (También trata de uno de los parientes cercanos del 8086, el 8088, de manera que, a menos que se indique lo contrario, todo lo que se diga para el 8086 es válido para ambos.) Mostraremos cómo trabaja el 8086, cómo puede acoplarse a otros chips y cómo sacarle el máximo rendimiento. El 8086 fue el primer chip de esta nueva generación en salir al mercado, por lo que cuenta con una gran mayoría de seguidores, tanto usuarios como fabricantes y diseñadores de productos de soporte. A esto hay que añadir que es el más barato de entre la gran mayoría de chips de 16 bits actualmente en el mercado.

Este libro se pensó para enseñar, de una manera esencialmente práctica, cómo utilizar el 8086 junto con toda su familia de chips de soporte, y sus «hermanos», el 186, 286 y 432. Estamos seguros que aquellos lectores que ya estén familiarizados con el antiguo 8080 de Intel querrán conocer todas las posibilidades de este nuevo 8086, y estamos seguros que no les será difícil, puesto que el 8086 es, en el fondo, una ampliación lógica del anterior. Este libro introductorio presenta una primera aproximación visual que cubre los conceptos básicos del 8086 y su familia. No es un manual de referencias, sino un libro pensado para principiantes que ya conocen algo del mundo de los microprocesadores de 8

bits, y quieren conocer la tecnología de los nuevos elementos de 16 bits.

¿QUE TIENEN DE PARTICULAR LOS MICROPROCESADORES DE 16 BITS?

Lo que hace a los nuevos microprocesadores de 16 bits especialmente interesantes es su posibilidad de tratar la información en múltiplos de 16 bits en vez de 8. Hoy en día, los nuevos procesadores son capaces de tratar datos de varios tamaños básicos, incluyendo 8 y 16 bits, pero los mecanismos esenciales de los procesadores de 16 bits son siempre el doble que los equivalentes de los procesadores de 8 bits. Además, dado que cada parte del dato puede ser dos veces mayor, el procesador ofrece una precisión mucho más alta. El número mayor representable con 8 bits es 255, mientras que con 16 bits puede llegarse a 65.535, un número ¡256 veces mayor! Tanto el rango de representación de números como la precisión aumenta considerablemente en los distintos formatos numéricos. Es interesante resaltar que los microprocesadores 8086/8088 todavía utilizan un juego de instrucciones «orientado a octetos *(bytes)*»; esto es, cada instrucción en lenguaje máquina del 8086 ocupa de 1 a 6 octetos de memoria. Las instrucciones del antiguo 8080 necesitaban de 1 a 3 octetos. La ampliación en el 8086 le permite poseer un juego de instrucciones más amplio y versátil. Al mismo tiempo, al trabajar con 16 bits, el procesador puede cargar las instrucciones a doble velocidad que el 8080. La capacidad de tratar los datos de 16 en 16 bits aumentará la velocidad del microprocesador en otros sentidos. Así, por ejemplo, un procesador de 16 bits puede enviar y recibir un dato de 16 bits en una única transferencia, mientras que el procesador de 8 bits necesitaría para lo mismo dos operaciones distintas. El software también gana eficiencia, al resultar más fácil representar y operar grandes números de 16 bits en los registros de la CPU. En un microprocesador de 8 bits se necesitan dos pasos para operar números de 16 bits, y el número de instrucciones a utilizar puede fácilmente ser cuatro veces mayor que las necesarias en un procesador de 16 bits.

De todas maneras, la posibilidad de tratar a la vez datos de tamaño doble es sólo una parte de la historia de los microprocesadores de 16 bits. Los fabricantes y diseñadores pretendieron aumentar el poder operativo más allá de lo que implica el doblar el tamaño de las palabras de datos, y para ello desarrollaron una circuitería más compleja en los chips. Un microprocesador, en el fondo, no es más que una computadora hecha de miles de transistores minúsculos (los transistores pueden imaginarse como pequeños amplificadores que controlan de forma precisa el flujo de señales eléctricas), introducidos en un minúsculo chip (pastilla de circuito integrado). Los transistores se *depositan* sobre finas planchas de cristal por medio de técnicas fotográficas de miniaturización. Las planchas de cristal se *tratan* convenientemente en un ambiente electro-químico, para ser posteriormente cortadas y encapsularse en chips.

El grado de miniaturización que puede conseguirse con estas técnicas determina la cantidad de transistores que podrán encap-

sularse en la computadora. Cuantos más transistores, más cosas podrá hacer el procesador. Para implementar el 8086 se necesitaban nuevas técnicas de integración que permitiesen disminuir el tamaño de los transistores, y se desarrolló un nuevo proceso de tratamiento del silicio, el llamado HMOS. La técnica HMOS permite introducir en un simple chip 70.000 transistores o más. Con tantos transistores se pueden incluir extensos circuitos hardware. Por ejemplo, se pudo incluir en el propio chip las instrucciones de multiplicación y división. Al no estar simuladas por software, estas operaciones se realizan en un tiempo mucho menor. Con tantos transistores se pudieron también crear sofisticadas estructuras de interrupción (algunas con circuitos de prioridad incorporados) e incorporarlas al chip. Esto supone un control mucho más eficiente de casi todos los dispositivos periféricos de entrada/salida con muy poco hardware adicional.

Hay otras características de los chips de 16 bits que los sitúan muy por encima de sus hermanos de 8 bits. Por ejemplo, la cantidad de memoria que puede utilizar un chip de 16 bits es enorme comparado con la de un microprocesador de 8 bits. El 8086/8088, por ejemplo, puede acceder a cerca de un millón de octetos de memoria de lectura/escritura, lo que contrasta con el máximo de memoria que puede utilizar un microprocesador de 8 bits, que suele ser de unos 64.000 octetos. Con tal cantidad de memoria, los programadores pueden desarrollar programas mucho más potentes y sofisticados. Con memorias de unos 256.000 octetos es ya posible utilizar sistemas operativos avanzados que soporten lo mejor de los productos desarrollados para computadoras potentes. El poseer gran memoria posibilita que varios usuarios trabajen simultáneamente, compartiendo la computadora. No es excesivo pensar en un «procesador de textos» de 1 megaocteto de memoria dando servicio a 16 usuarios a la vez. Cada usuario tendría su terminal con su propio 8086 incorporado, comunicado con la computadora central.

Pero quizás el aspecto más relevante y misterioso de los nuevos microprocesadores de 16 bits es la forma en la que sus diseñadores han distribuido sus funciones inteligentes. Los nuevos chips de 16 bits poseen mecanismos que soportan estructuras de cálculo más potentes que la de un solo procesador realizando una sola tarea. Mientras que los antiguos microprocesadores realizaban los cálculos en un sistema, los nuevos micros de 16 bits distribuyen las funciones a realizar en subfunciones que se ejecutan en chips opcionales especialmente diseñados para ello. Estos chips no son obligatorios para las funciones usuales, pero están disponibles en caso de que se quieran utilizar. Hoy en día hay chips especiales que realizan operaciones en coma flotante, funciones trigonométricas, e incluso funciones especiales de E/S sin necesidad del procesador central.

Estos procesadores suplementarios, o procesadores paralelos, son algo más que simples chips de soporte. Son realmente microprocesadores completos dedicados a tareas especiales que antiguamente requerían placas completas para contenerlos. Un

ejemplo es el Chip Matemático de Intel 8087. El 8087 es un potente microprocesador, con su propio lenguaje de programación, que puede realizar operaciones de alta precisión en un tiempo increíblemente corto. Mientras que una raíz cuadrada en doble precisión (53 bits), ejecutada en un microprocesador de 16 bits por software, puede necesitar fácilmente 20 mili-segundos, con un chip de este estilo se ejecuta en menos de 36 micro-segundos, ¡más de 500 veces más rápidamente! (Por supuesto, esta razón tan buena no se mantiene en todas las operaciones normales.) Este chip elimina la necesidad de utilizar rutinas de emulación, normalmente complejas y que requieren bastante memoria. El uso de este chip con un lenguaje de programación como el BASIC o Pascal significa no sólo una mayor rapidez de cálculo, sino un ahorro en la cantidad de memoria utilizada, siendo de esta forma más fiable y barato.

Estos chips matemáticos, o procesadores paralelos, no se limitan a funciones científicas, o a extensiones de lenguajes de alto nivel. Por ejemplo, es posible diseñar un controlador industrial barato y de alto rendimiento, con un chip matemático y en 8088. Esta combinación permite efectuar operaciones complejas en tiempo real. El registro digital, el análisis espectral, la música sintética, el reconocimiento de voz y las comunicaciones son algunas aplicaciones todavía sin explorar de estas nuevas máquinas.

Además de dotar al microprocesador de 16 bits de posibilidades de conexión creativas, los diseñadores han incluido en él ciertos «trucos» atractivos. Con la disminución progresiva del precio del hardware, la programación de computadoras se está convirtiendo en un trabajo complejo y lento; en el aspecto más caro de cualquier producto basado en micros, tanto sea para un controlador de procesos industriales, un aparato orientado al mundo de los negocios o una computadora personal. Se han incorporado funciones al nuevo chip de 16 bits que permiten «mover» los programas fácilmente por memoria, cuestión importante si se desea un software potente. Esta posibilidad permite, por ejemplo, que un programa se auto-reconfigure sobre la marcha, creando versiones limitadas por la cantidad de memoria disponible en cada momento. Tanto es así que, al ser fácilmente reubicables los programas, los programas especialmente diseñados para el microprocesador de 16 bits pueden trabajar en armonía con otros programas de otros fabricantes.

Otra área donde el impacto del nuevo procesador de 16 bits va a ser importante es en la detección y tratamiento de errores. El tratamiento de errores es un área demasiado olvidada por la industria de microcomputadoras. Un programa que no trate correctamente los errores puede ocasionar graves problemas. Así, si la detección de errores es pobre, pueden quedar errores sin detectar que conviertan el resultado cuanto menos en sospechoso. O, si la recuperación de errores es pobre, el programa puede quedar «bloqueado», sin capacidad de reanudación a no ser destruyendo parte del trabajo realizado. Por otro lado, si el

proceso de recuperación es bueno y ocurre un error, el programa mismo será capaz de arreglarlo automáticamente y continuar con el proceso. Todas las nuevas unidades de 16 bits poseen algoritmos potentes de detección y recuperación de errores. Especialmente el nuevo procesador matemático. Por ejemplo, si ocurre un error durante la instrucción de división, como podría ser un desbordamiento o una división por cero, el microprocesador lo detecta y provoca una interrupción del proceso. El programa puede tratar convenientemente esta interrupción, sea enviando un mensaje de aviso por consola, sea devolviendo el control al programa, insertando valores más apropiados o lo que se desee. Con estos mecanismos de tratamiento de errores es tarea fácil el diseñar programas que no se bloqueen nunca y que no paralicen al sistema.

Con este grado de sofisticación, parece claro que pronto, muchos sistemas operativos y lenguajes de alto nivel como el lenguaje C, UNIX, Pascal, Forth y Ada, estarán disponibles para las computadoras basadas en los maravillosos chips de 16 bits, a precios razonables.

El 8086 provee potencial para todos aquellos productos de consumo que requieran de su habilidad para ejecutar programas complicados de una forma rápida y eficiente. Cualquier producto que en este momento utilice algún chip procesador es susceptible de mejora con el nuevo microprocesador de 16 bits. Los primeros candidatos claros a incorporar el 8086 son tal vez las computadoras personales de alto rendimiento (tales como pequeñas computadoras de empresa con tratamiento de gráficos), los videojuegos, los automóviles, los equipos de cocina, máquinas de escribir, contestadores automáticos y radios.

Dada la potencia de los chips de 16 bits, las comunicaciones orales constituirán una parte primordial de los productos informáticos de los ochenta. Así por ejemplo, las computadoras personales podrán pronto programarse de manera que puedan preguntar datos y escuchar las respuestas que se les den. Los terminales punto de venta no tendrán teclas, sino simplemente una entrada para monedas, un altavoz y un micrófono. El espectáculo de ver personas hablando a estas cajas inteligentes dejará de ser un acontecimiento.

Dado que el 8086 no resulta mucho más caro que el 8080, y que puede que en el futuro resulte incluso más barato, se puede utilizar en todos aquellos lugares en los que el peso, coste y una inteligencia superior sean factores críticos, como en un avión, o, desgraciadamente, en ciertos tipos de armas. Con los subsistemas de un avión controlados independientemente por varios 8086 en modo *esclavo,* todo el avión puede operar de forma más precisa, rápida y exacta. El procesador matemático realizaría los cálculos de navegación, afianzando el sistema de piloto automático. Resulta curioso pensar que, con suficientes procesadores de este tipo, un avión de combate podría tener tanta inteligencia por él mismo, que el piloto podría quedarse en tierra, en una carlinga simulada, ¡combatiendo por computadora!

La primera aplicación del 8086 será, sin duda, el mercado de las computadoras personales, un mercado cada vez más competitivo. En agosto del 81, IBM anunció su primera computadora personal. Además de convulsionar el mercado haciendo una computadora que podía utilizar gran parte del software para procesadores de 8 bits existente en el mercado, los diseñadores de IBM incorporaron el potente 8088. Internamente, el 8088 y el 8086 son iguales. Externamente, el 8086 utiliza un bus de 8 bits que le permite conectar las memorias y unidades de E/S normales de 8 bits. Con el IBM usando el 8088, sus computadoras personales se convertirán sin duda en unas de las más potentes del mercado. Gracias al 8088, IBM podrá desarrollar y utilizar software extremadamente potente. Por otra parte, la computadora personal de IBM reserva un zócalo para el procesador numérico 8087. Con él, y el software desarrollado a tal efecto, la computadora personal de IBM se convertirá en un perfecto «mago» de las matemáticas.

No es probable que el 8086 destaque en el campo de los videojuegos baratos, pero sí puede tener un papel primordial en el área de los contestadores automáticos. Con el 8086, un procesador numérico y un procesador de E/S, los mensajes recibidos se pueden guardar en una memoria de lectura-escritura (RAM), directamente dentro de la máquina, en vez de en una cinta. La potente estructura del lenguaje del 8086 convertirá al contestador automático en algo más que un simple sistema de captura de mensajes, pudiendo incluir programas «recordatorio» de citas e incorporar las posibilidades del «correo electrónico».

Otro lugar en el que probablemente se utilizará el 8086 junto con el teléfono es en las centralitas privadas digitales (PBX). La PBX es una centralita que permite conectar todos los teléfonos de una empresa con el exterior, dirigidos y controlados por uno o más empleados. Las PBX convierten cada teléfono del sistema en una estación inteligente. En la mayoría de los casos, cada PBX tiene varios empleados que se encargan de gestionar la recepción de llamadas y las peticiones de conferencia. El 8086 permitirá sustituir los empleados por una computadora inteligente junto con un sintetizador de voz capaz de contestar a las llamadas a la compañía con un «Buenas tardes, aquí la compañía XYZ, ¿con quién desea hablar, por favor?», y capaz de actuar en consecuencia cuando le contesten «Con el señor Harvey Mudd, por favor». Además, la computadora siempre contestará el mensaje de una forma clara y precisa que el interlocutor pueda entender (al contrario de algunas telefonistas, que después de 3.000 llamadas empiezan a hablar entre dientes). Una PBX automática podría incluso conectar entre sí llamadas exteriores, cosa que no puede hacer un recepcionista u operador humano.

Los microprocesadores de 16 bits tendrán sin duda una gran influencia en la forma de funcionamiento de las máquinas actuales, y en su forma de relación con los seres humanos. Confiamos en que estos dispositivos contribuirán a conseguir un mundo mejor, en vez de complicarlo todavía más.

BREVE REPASO AL 8086/8088

Puesto que esto es un capítulo introductorio, daremos un repaso a las características básicas del 8086 y del 8088. En los capítulos siguientes se cubrirá con más detalle la familia completa de chips.

Los microprocesadores de 16 bits 8086 y 8088 son una extensión lógica del popular 8080. Internamente son iguales, pero el 8088 está diseñado para trabajar con un bus de 8 bits, siendo de esta manera compatible con la mayoría de buses del mismo tamaño. El 8086, por el contrario, se conecta a un bus de 16 bits. El 6 del 86 indica el bus de 16 bits; el 8 del 88 significa que el bus en este caso es de 8 bits. Ambos números se refieren a la anchura física del bus de datos. Internamente, ambos poseen el mismo juego de instrucciones y el mismo tamaño de datos.

Los dos, el 8086 y el 8088, utilizan el concepto de las *colas de instrucciones* para aumentar la velocidad de proceso. Dentro del propio chip hay un área especial a la que se da el nombre de *cola de instrucciones*, que guarda los octetos de la instrucción. Cuando la computadora está preparada para obtener la siguiente instrucción, no necesita cargar los octetos desde memoria, sino que toma la instrucción siguiente de la cola antes citada. De esta forma, el bus de datos y el de direcciones no presentan períodos punta de utilización como es común en los buses de los dispositivos de 8 bits, que necesitan estar continuamente accediendo a memoria. Los buses de las computadoras son algo así como grandes autopistas, con períodos tranquilos y horas en las que se colapsan totalmente. El secreto del 8086 es tener un sistema de colas que tiende a eliminar esas horas punta, desviando el tráfico a períodos más tranquilos, y controlar así más eficazmente el flujo de tráfico. Utilizando mejor el bus se consigue aumentar el *ancho de banda*

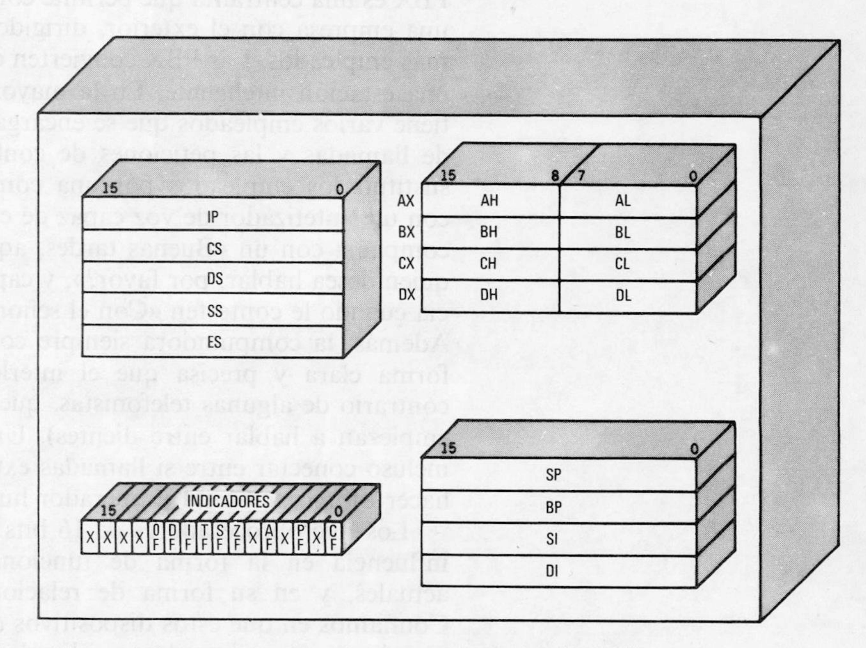


Figura 1.2: Estructura del microprocesador de 16 bits 8086.

del bus, y aumentar el número de instrucciones posibles por segundo, al lograr que el bus esté durante más tiempo disponible para el resto de los dispositivos. La cola del 8086 es de 6 octetos, y la del 8088 de 4. El tamaño de la cola tiene relación con el mayor rendimiento del 8086 sobre el 8088.

El 8086 puede acceder a 1 megaocteto de memoria de lectura-escritura (2^{20} octetos). Sin embargo, utiliza un esquema de direccionamiento de memoria llamado de *segmentación,* en el cual ciertos *registros de segmentación* suministran una dirección base, que se añade automáticamente a cada dirección de 16 bits que define el usuario. Aunque hay cuatro registros de segmentación en el 8086, la dirección base posible (dada por los registros de segmentación) puede emplazarse a intervalos regulares de 16 octetos a lo largo de todo el megaocteto de memoria direccionable. En los capítulos 2 y 3 se explica cómo hacerlo.

Parte de la dirección y todo el bus de datos están multiplexados en 16 terminales. Los 4 bits de dirección restantes se corresponden con los 4 terminales de dirección adicionales que también se utilizan para el estado. Se requiere un reloj externo y un controlador de bus también externo que se utiliza para demultiplexar el bus de direcciones/datos. La figura 1.2 muestra la estructura de los registros internos del 8086/8088. Los registros sombreados representan el subconjunto del 8080.

El 8080 tiene una estructura de interrupciones muy potente. Casi todos los microprocesadores de 16 bits necesitan chips externos adicionales para conseguir unas buenas interrupciones. En el 8080 se utilizan cerca de un millón de octetos para gestionar los 256 vectores de interrupciones. El 8086 realiza las operaciones de E/S en un espacio físicamente separado de memoria llamado *espacio de E/S* de hasta 64 K octetos de longitud. Para el uso de procesadores paralelos, hay un terminal (pin o patilla) especial de entrada, TEST, que permite al 8086 conocer cuando el procesador paralelo ha completado su tarea. Cuando se encuentra una instrucción WAIT, el 8086 para y espera a que el procesador externo, o cualquier otro dispositivo hardware, le indique que continúe, activando el terminal de TEST.

El 8086 contiene estructuras para código independiente de la dirección y de la gestión de pilas (no hay que alarmarse, en seguida explicaremos estos términos). Tiene instrucciones aritméticas para números de 16 bits con signo, incluyendo multiplicación y división, gestión de los bits superiores, y operaciones con cadenas interrumpibles, que examinan el texto y paran cuando se produce o no una igualdad. Esta es una gran ventaja para los procesadores de texto al permitir unas operaciones muy rápidas.

El 8086 tiene unos tipos de registros nuevos además de los usuales del 8080. En primer lugar los tres registros pares del 8080 (HL, BC y DE) reciben ahora los nombres de BX, CX y DX respectivamente. Pueden tratarse indistintamente como pares de registros de 8 bits, o como registros de 16 bits respecto a casi todas las operaciones de 16 bits. El registro acumulador del 8080, A, es ahora de 16 bits y recibe el nombre de registro AX; pero su

mitad menos significativa puede seguir utilizándose como registro acumulador de 8 bits. Hay cuatro registros nuevos de 16 bits de gran interés. Se trata de los registros de segmentación mostrados en la figura 1.2 con las siglas CS, DS, SS y ES. Se utilizan en la *segmentación* del 8086. A través de ellos se puede decir a la computadora, separada y dinámicamente, la dirección de un programa, dato, o pila, dentro del megaocteto de memoria.

Hay todavía cuatro registros más de 16 bits: el puntero de pila (como en el 8080), el puntero base y dos registros índice —el registro índice fuente y el de destino.

El 8086 es muy rápido. Con un reloj de 5 MHz., se tarda dos microsegundos en cargar el acumulador desde cualquier posición de memoria dentro del megaocteto. A 8 MHz., son necesarios sólo 1,25 microsegundos. Pronto estarán disponibles memorias aún más rápidas que permitan realizar esta operación ¡en menos de un microsegundo! El 8086 incluye 29.000 transistores en un área de 225.000. Es tan sólo un 27 por 100 mayor que el 8080, de modo que su coste resulta muy reducido.

Junto con el 8086 hay dos procesadores paralelos disponibles. El Procesador de Datos Numérico 8087 ofrece al 8086 operaciones muy rápidas en coma flotante. Está diseñado para recubrir el bus de datos/direcciones locales del 8086, de forma que se puede acoplar con pequeños cambios y un mínimo de hardware extra. El 8087 controla el flujo de instrucciones del 8086 localizando sus propias instrucciones y ejecutándolas sin la ayuda del 8086. El segundo procesador paralelo es el Procesador de Entrada/Salida 8089. Se trata éste de un chip diseñado para una gestión eficiente de los movimientos de bloques de datos. Tiene dos canales y puede solapar entradas y salidas fácilmente. Se solapa también con el bus local, y tiene su propio juego de instrucciones. Estos chips, junto con el 8086, son una sólida base para el diseño de computadoras realmente potentes.

SOFTWARE DEL 8086

Este es un tema realmente importante. Todos hemos oído alguna vez la frase «sin software un microprocesador no es nada». A lo que se refiere es que para utilizar al completo un microprocesador, se deben tener abundantes programas disponibles que puedan trabajar con la computadora. Hoy en día, las computadoras basadas en microprocesadores de 8 bits tienen un vasto número de *programas de aplicación* disponibles. Para el popular 8080 hay miles de vendedores que ofrecen los más variados programas de aplicación. Pero el nuevo microprocesador de 16 bits tiene sus propias y particulares instrucciones (más potentes). No se pueden tomar simplemente un programa desarrollado para un microprocesador de 8 bits y trasladarlo al de 16. Es necesario pagar a un programador para que cree una nueva versión de la aplicación especial para el microprocesador de 16 bits. La nueva computadora es más potente y mejor, pero ha de pasar cierto tiempo hasta que comiencen a aparecer programas de aplicaciones para él. Durante ese tiempo, el usuario se ve trabado por una computadora potente, pero casi sin software disponible. Afortu-

nadamente, esto no es un problema para el 8086, ya que hay una correspondencia muy fuerte entre los registros e instrucciones del antiguo 8080 y un subconjunto de los registros e instrucciones del 8086. Por supuesto, los nombres se han cambiado «para proteger al inocente». Hay tan sólo unos pocos programas «grandes» en lenguaje de bajo nivel, que es necesario traducir o desarrollar de nuevo, pero una vez conseguido, hay un gran número (toneladas) de programas de aplicación en lenguajes de alto nivel, inicialmente desarrollados para el 8080 que pueden trabajar perfectamente en el 8086.

Para facilitar todavía más esta conversión, existen programas especiales traductores que pasan el código del 8080 a código del 8086. Los diseñadores del 8086 y otros, incluso nosotros mismos, han desarrollado programas de este tipo. A lo mejor el lector, cuando haya acabado este libro, le entran ganas de escribir alguno más. La traducción es un proceso casi automático, aunque no totalmente. El programador debe todavía *retocar* el código para inicializar los registros de segmentación, tratar algunas incompatibilidades entre ensambladores y arreglar algún defecto «furtivo» de la programación a nivel máquina del 8080. La traducción mecanizada tampoco produce un código óptimo, pero a cambio ahorra tiempo en el proceso de conversión. Con todo esto, las compañías que hoy en día ofrecen software para el 8080, no hay duda que no tardarán en vender versiones para el 8086.

Un punto esencial en el diseño de software es el problema de los sistemas operativos. Este es un tema extraño aunque sumamente importante. Cualquier programa que se quiera ejecutar en el 8086 requiere otro programa, llamado *sistema operativo*. El sistema operativo es algo así como un «programa madre» que revisa las peticiones de los programas de los usuarios y ayuda a cada programa particular a ejecutarse, suministrándole vías de acceso a los diferentes dispositivos de la computadora, como el teclado, discos flexibles e impresora. El sistema operativo permite que los programas particulares sean relativamente independientes de la configuración de la computadora abriendo de esta manera el amplio mercado de los programas de aplicación. El problema aparece por el hecho de que no hay un único sistema operativo disponible para una computadora de 16 bits. Al escoger uno u otro sistema operativo, se está restringiendo el rango de software que se podrá adquirir. Para complicar más las cosas, existen lo que se llaman sistemas operativos «low-end» (inferior-de una gama) y «high-end» (superior-de una gama). Por ejemplo, CP/M es un típico sistema operativo «low-end». CP/M es uno de los sistemas operativos más populares (para más detalles, consultar *CP/M Primer,* Steve Murtha, Mitchell Waite; Howard W. Sams & Co., Inc., Indianápolis, Indiana). CP/M resulta barato y muy simple de usar, pero también tiene algunos inconvenientes. Sistemas operativos más sofisticados, «high-end», como el OASIS o el UNIX son más caros, pero ofrecen una flexibilidad increíble. Intel también tiene su propio sistema operativo, el ISI-II. De todas maneras, lo cierto es que el sistema operativo que se elija

determina el rango de programas estándar que se podrán adquirir.

Cada vez hay más sistemas operativos diseñados especialmente para el 8086 y 8088 en el mercado. Uno de los primeros sistemas operativos que aparecieron para el 8086 fue el CP/M-86 de Digital Research Inc., el fabricante del CP/M-80, el sistema operativo más popular mundialmente. CP/M-86 se parece bastante al CP/M-80, pero utiliza algunas características especiales de los chips de 16 bits. Su utilización presenta algún problema. Por ejemplo, sus procedimientos de recuperación de errores son pobres y deben modificarse particularmente para cada computadora. Otro sistema operativo para el 8086 es el MS-DOS, desarrollado por Microsoft. El MS-DOS está escrito en el lenguaje de programación C, y puede recompilarse para trabajar en el Z8000 y en el 68000, y, por lo tanto, en casi cualquier microprocesador de 16 bits. Es compatible con el sistema operativo XENIX, muy similar al UNIX, de la misma compañía. Se ha desarrollado parcialmente después que CP/M-80 y emula cualquier llamada de su sistema. Los programas para 8 bits que trabajan en CP/M pueden pasarse a código 8086, y las llamada al CP/M, naturalmente, se conectan bajo en MS-DOS. Utiliza dispositivos de E/S independientes, de forma que cada dispositivo actúa como un fichero; se abren y cierran, se leen y se escriben. Cualquier programa MS-DOS puede decir «Manda esto a la pantalla», e independientemente de las características hardware de la pantalla, el mensaje aparece correctamente. El MS-DOS es reubicable, permitiendo a los programas utilizar la segmentación del 8086.

Quizás la característica más importante del MS-DOS es el tratamiento de gráficos independientemente de los dispositivos, largamente esperado en la industria de microcomputadoras. El MS-DOS utiliza los estándares ATT para transmisión de teletexto, y un formato mejorado del Telidon, el llamado Protocolo de Nivel Presentación (PLP). Dado que el ATT se utiliza en la red telefónica americana, un sistema operativo que contenga órdenes que interpreten PLP tiene un potencial tremendo. Permite a los programadores escribir los gráficos sin pensar en una computadora particular. Recordemos que, aunque el Apple tiene una capacidad gráfica increíble, los programadores necesitan «años» para adaptar su código, debido al extraño diseño hardware. Con el PLP, el programador no debe preocuparse del hardware. El diseñador de hardware diseña un visualizador que puede interpretar las órdenes del PLP. Estas órdenes son instrucciones para figuras geométricas, etc.

El nuevo computador personal de IBM corre sobre una versión del MS-DOS llamada Computadora Personal IBM-DOS. Los expertos predicen que el MS-DOS distanciará pronto al CP/M-86, convirtiéndose en el sistema operativo más popular para microcomputadoras de 16 bits. En el último capítulo del libro se da una lista que describe las posibilidades software en el momento de escribir este libro.

La lista aumenta rápidamente, así que es recomendable

investigar el mercado del momento antes de tomar ninguna decisión. La lista incluye sistemas operativos, lenguajes de alto nivel como BASIC, FORTRAN IV y Pascal, herramientas como macro-ensambladores, ensambladores cruzados, cargadores encadenadores (editores de enlaces), editores e, incluso, ¡una gama completa de divertidos y excitantes video-juegos! Sin duda, hay todo un frenesí de actividad alrededor de todo lo relacionado con los productos software para el 8086. Y el rendimiento de estas nuevas versiones será muy superior a las diseñadas para las máquinas de 8 bits, hasta el punto que las antiguas versiones quedarán completamente obsoletas.

ORGANIZACION DEL LIBRO

Este libro está organizado en ocho capítulos. Se ha escrito de manera que pueda leerse secuencialmente desde el capítulo 1 al 8, aunque se pueden saltar algunos capítulos siempre que se tengan en cuenta los siguientes puntos:

Este primer capítulo, «Perspectivas», es una introducción a los microprocesadores de 16 bits, especialmente al 8086.

El capítulo 2, «Conceptos básicos de los microprocesadores de 16 bits», presenta para el lector no iniciado, una introducción completa al mundo de los microprocesadores de 8 y 16 bits, y a los nuevos conceptos asociados con el nuevo procesador de 16 bits. En este capítulo se introduce la nomenclatura y la arquitectura de una microcomputadora basada en microprocesadores, y se pasa revista no sólo al 8086/8088, sino a toda la familia 8000. A continuación, el capítulo introduce algunos conceptos básicos necesarios para conocer y apreciar las máquinas de 16 bits. Cubre los tipos de datos y números, desde bits y cuartetos hasta cadenas y punteros. Se cubre también la organización física de la memoria, su gestión, el procesamiento (tratamiento) en paralelo y las colas de instrucciones. El capítulo finaliza con una descripción del software utilizado durante el desarrollo de los programas que aparecen en este libro.

El capítulo 3, «Los procesadores de propósito general 8086/8088», cubre las entradas y salidas de estos chips específicos, incluyendo las similitudes entre el 8086 y el 8088, sus principales características, requerimientos de alimentación, arquitectura tabular (pipeline), señales y terminales de salida, etc. Se introduce el juego de instrucciones agrupadas por funciones como transferencias de datos, aritmética binaria, gestión de cadenas, etc. Finalmente, se presentan unos cuantos programas-ejemplo para que el lector los examine. Este es un capítulo que, a nuestro entender, debe leerse.

El capítulo 4, «El Procesador de Datos Numérico 8087» es un capítulo muy interesante que muestra cómo trabaja el procesador numérico. Cubre los conceptos de representación de números en coma flotante, y explica cómo trabaja el NDP 8087 como procesador paralelo, detallando cómo queda ampliado el software, la arquitectura, el hardware y la comercialización del 8086/8088. Se estudian los tipos de datos del 8087 (clases de números con los que puede operar y su magnitud), sus pilas, su

potente juego de instrucciones y la gestión de condiciones excepcionales. El capítulo concluye con algunos programas-ejemplo.

El capítulo 5, «El Procesador de E/S 8089», explica este entretenido e inteligente dispositivo DMA de alta velocidad, y como utilizarlo eficientemente. Este capítulo es un mini-libro en sí. De nuevo, el capítulo finaliza con programas-ejemplo sobre como utilizar el IOP 8089. Si no se está interesado en este dispositivo, se puede saltar tranquilamente el capítulo.

El capítulo 6, «Los chips de soporte 8086/8088», resume la familia de chips 8200. Una familia de chips de soporte no es más que un grupo de chips que pueden operar conjuntamente en ayuda del procesador central. Una buena comprensión de estas familias permite apreciar mejor las posibilidades de configuración de los sistemas basados en el 8086/8088.

El capítulo 7, «Trabajando con el 8086/8088», presenta una colección de programas de aplicación sobre CP/M, desarrollados como ilustración de la programación del 8086/8088. Son programas cortos y fáciles de entender. Cada programa introduce un grupo de instrucciones. Por ejemplo, se empieza por las instrucciones de transferencia de datos viendo cómo utilizarlas para mover los caracteres «HI» en la pantalla. A continuación se ven las instrucciones de bucles, estudiando como realizar cuentas multidígito en la pantalla (combinación de dos procesos), etc. Para programadores en potencia del 8086/8088, éste es un capítulo muy interesante. Aquellos que no deseen conocer el chip de forma tan detallada, pueden omitir el capítulo.

El capítulo 8, «Estado actual», cubre la evolución del hardware y software del 8086, el desarrollo del chip, la computadora personal IBM, y da una lista parcial del equipamiento y software disponible hoy en día, con los nombres y direcciones de los fabricantes y vendedores de productos basados en el 8086.

En el libro se han utilizado bastantes dibujos tridimensionales y figuras en un intento de clarificar al máximo los conceptos. Mientras lea este libro, recuerde que, entendiendo el 8086, estará en una posición óptima para programar computadoras como la computadora personal de IBM.

Creemos que muy pronto habrá demanda de buenos programadores para este tipo de máquinas, y se ofrecerán buenos salarios. Dicho de otra manera, el mercado de los 16 bits está abierto, totalmente virgen, así que, ¿por qué no empieza ya a leer el siguiente capítulo?

Capítulo 2

Conceptos básicos de los microprocesadores de 16 bits

En este capítulo estudiaremos algunos conceptos básicos relativos a los modernos microprocesadores de 16 bits. Incluiremos temas como la organización de una microcomputadora típica moderna; un repaso a la extensa familia Intel de microcomputadoras y chips microprocesadores; algunos conceptos particulares de los nuevos procesadores de 16 bits tales como la segmentación, gestión de memoria y multiproceso; varios tipos comunes de datos, y una somera introducción al lenguaje de programación Ensamblador (e incluso a niveles más bajos).

Muchos de los conceptos básicos introducidos en este capítulo se tratarán con más profundidad en otras partes del libro, cuando estudiemos cada uno de los chips. Asimismo, en capítulos subsiguientes introduciremos nuevos conceptos básicos conforme vayan siendo necesarios.

MICROCOMPUTADORAS: DE AFUERA A ADENTRO

La gran mayoría de la gente está ya familiarizada con los dispositivos externos que un microcomputador lleva acoplados. Normalmente tendremos un teclado, una pantalla, un par de unidades de discos flexibles y una impresora. Puede haber también otros dispositivos como registradores gráficos (plotters), paletas de control (joystick) y digitalizadores (ver figura 2.1). Todos estos sistemas están conectados por medio de un cable a una «caja» que alberga a la computadora en sí. Esta caja suele

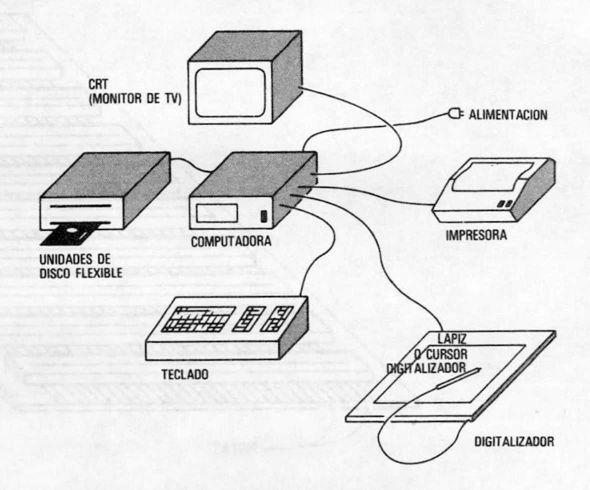


Figura 2.1: Sistema computador típico.

incluir también un teclado, una pantalla y/o tal vez alguna unidad de discos flexibles, pero, incluso en estos casos, podemos pensar en un *sistema computador* como en un *computador central* con sistemas periféricos acoplados.

Arquitectura basada en buses

Echemos un vistazo al interior de la computadora. Normalmente se compone de un bus principal con varios dispositivos conectados a ella. El bus aparece como un racimo de conductores eléctricos en paralelo que atraviesa la computadora. El famoso bus S-100 de algunas computadoras está ubicado en una placa, llamada *placa matriz,* que normalmente se encuentra en la parte inferior de la computadora. Las cien líneas de señal del bus corren paralelamente de un extremo a otro de la placa. Físicamente, las señales son transportadas por una película de metal depositada sobre una base aislante formando pistas. A lo largo de la placa y colocados perpendicularmente a las pistas hay de 5 a 20 conectores, conectados a todas las líneas de señal del bus (pistas), y que permiten enchufar diversas placas al S-100. Dichas placas contienen funciones tan importantes como la memoria, la unidad central de proceso (CPU: Central Process Unit) y la interfaz de entrada/salida. Como puede verse en la figura 2.2, los conectores son paralelos entre sí pero perpendiculares a las pistas de señal del bus S-100. Este es un ejemplo muy sencillo del principio de

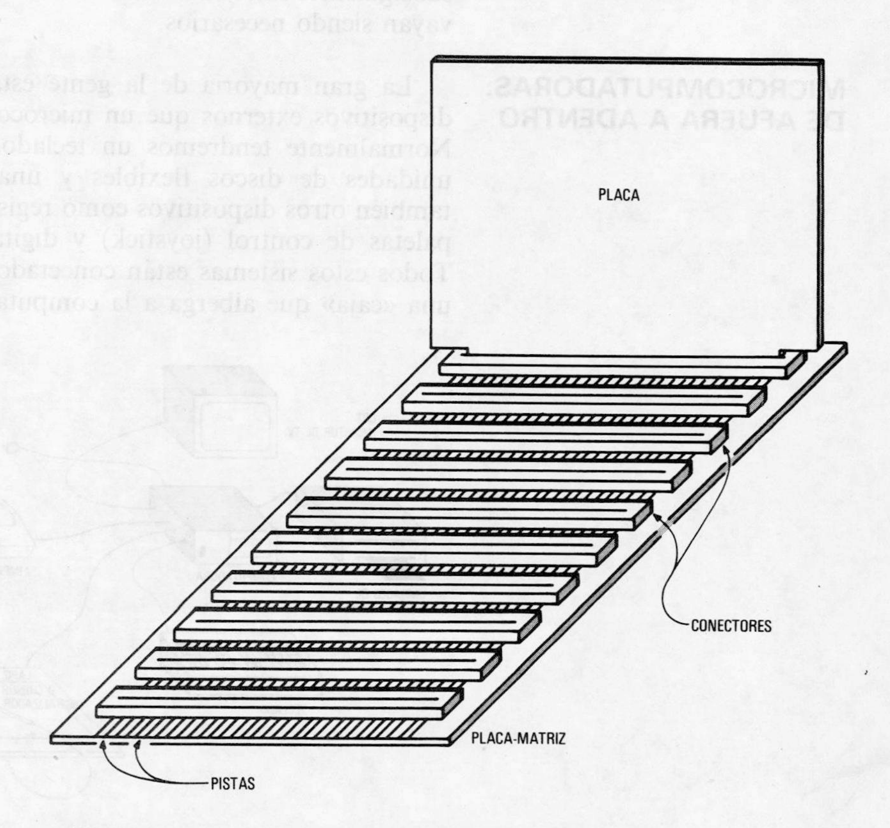


Figura 2.2: Un bus típico-placa madre del bus S-100.

ortogonalidad (perpendicularidad) mediante el cual se consigue que dos cantidades o cualidades interaccionen de forma totalmente uniforme. En este caso cada conector tiene acceso a exactamente el mismo número de líneas de señal del bus, de manera que las placas del S-100 pueden enchufarse a los conectores en cualquier orden, si bien es cierto que algunas disposiciones consiguen además reducir niveles de ruido del sistema.

Ni el bus S-100 ni los conductores eléctricos tienen *elementos activos;* esto es, normalmente no contiene transistores ni circuitos integrados, aunque algunos fabricantes incluyen unos pocos de ellos en lo que llaman *circuitería terminal,* cuya misión es minimizar el nivel de ruido a lo largo del bus. En algunos casos, parte de la fuente de alimentación está en la placa-matriz. Sin embargo, exceptuando la circuitería terminal y, desde luego, la fuente de alimentación principal, todos los elementos activos se localizan en las placas enchufadas a los conectores del bus S-100 (la figura 2.3 muestra un diagrama de bloques de un sistema S-100 típico). En computadoras que no poseen el bus S-100, suele existir una placa en el circuito principal que contiene todos los componentes básicos de la computadora, y, a menudo, la placa-matriz que alberga el bus se conecta a dicha placa principal. Las funciones de la placa principal y la placa-matriz se pueden encontrar combinadas en una única placa. A pesar de todas estas diferencias, resulta muy conveniente adoptar el *modelo lógico* de un computador como el de un *bus,* al cual se la han conectado diversos dispositivos.

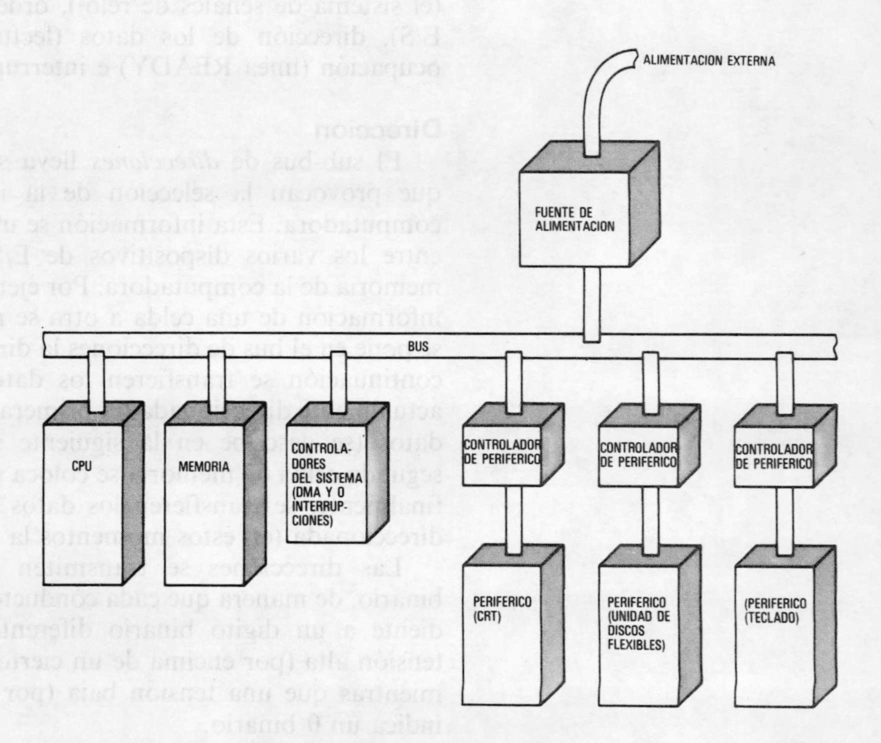


Figura 2.3: Diagrama de bloques de un sistema basado en buses.

El bus principal de una computadora se divide en varios sub-buses: 1) de alimentación, 2) de control, 3) de direcciones y 4) de datos.

Alimentación

El sub-bus de *alimentación* hace llegar la corriente proveniente de la fuente de alimentación a los distintos componentes de la computadora. El sistema 8080 necesitaba un sub-bus de alimentación de varias líneas debido a que el chip de CPU requería tres tensiones de alimentación diferentes, más una línea para tierra. Puesto que los nuevos chips (tanto microprocesadores como chips de soporte) sólo necesitan una tensión de alimentación (corriente continua, +5 voltios), este sub-bus está formado por únicamente dos cables: el de +5 voltios y el de tierra. Las líneas de alimentación de ± 18 voltios de los modelos anteriores no son necesarios en los actuales.

A veces se envía una señal de tensión más alta, con una ligera *ondulación* o «rizado» de 60 ó 120 hertzs (Hz), de manera que cada placa rectifica la onda y disminuye el voltaje, consiguiendo una corriente continua local de +5 voltios. De esta manera se consigue reducir el ruido, disminuir las *caídas de tensión* a lo largo del bus de alimentación y rebajar las señales parásitas entre las placas y el bus. Y, además, se reduce el precio del equipo.

Control

El sub-bus de *control* lleva información sobre la temporización (el sistema de señales de reloj), órdenes (memoria o acceso a la E/S), dirección de los datos (lectura o escritura), señales de ocupación (línea READY) e interrupciones.

Dirección

El sub-bus de *direcciones* lleva señales de control especiales que provocan la selección de la información a través de la computadora. Esta información se utiliza para distinguir a la vez entre los varios dispositivos de E/S y las miles de celdas de memoria de la computadora. Por ejemplo, cuando se desea mover información de una celda a otra se realizan los siguientes pasos: se pone en el bus de direcciones la dirección de la primera celda; a continuación se transfieren los datos de la celda de memoria actualmente direccionada (la primera celda de memoria) al bus de datos (se describe en la siguiente sección); la dirección de la segunda celda de memoria se coloca sobre el bus de direcciones, y finalmente, se transfieren los datos del bus de datos a la celda direccionada (en estos momentos la segunda celda de memoria).

Las direcciones se transmiten por el bus codificadas en binario, de manera que cada conductor lleva una señal correspondiente a un dígito binario diferente (bit: BInary digiT). Una tensión alta (por encima de un cierto límite) indica un 1 binario, mientras que una tensión baja (por debajo de un cierto límite) indica un 0 binario.

Debido a las pérdidas y otros tipos de imprecisiones del

sistema los dos estados sí/no, o los valores lógicos o binarios 1/0 vienen representados por *rangos* de tensión.

De acuerdo con las reglas de la aritmética binaria, con n dígitos binarios pueden representarse 2^n números binarios distintos. Los primeros procesadores de 8 bits como el 8088, 8085 o Z80 tenían 16 líneas de señal, y podían por tanto producir $2^{16}=65.536$ direcciones distintas. Esta cantidad parecía más que suficiente para la mayoría de las aplicaciones de microcomputadoras; sin embargo, con la continua reducción del coste de la memoria, hoy en día una microcomputadora de tamaño medio puede utilizar efectivamente 100.000 celdas de memoria, necesitando una dirección distinta para cada una de ellas. Al mismo tiempo, los lenguajes actuales de alto nivel, tales como Pascal o Ada, requieren también grandes cantidades de memoria de 64 K para arriba. El IEEE estándar propuesto para el bus S-100 especifica 24 líneas de señal para la dirección, dando por tanto un rango de direccionamiento de

$2^{24}=16.777.216$

¡Más de 16 millones de direcciones distintas! ¡Con esta cantidad de memoria podríamos guardar unas 16 copias de la Biblia!

Datos

El sub-bus de *datos* transporta la información a través de la computadora. Tanto en los modelos basados en el 8080, 8085 y Z80, así como en los más modernos basados en el 8088, el bus de datos tiene ocho conectores, y es capaz de transportar ocho señales en paralelo. Esto significa que el bus de datos puede llevar unidades de información de ocho dígitos binarios, sólo una unidad cada vez. Con estos 8 bits, cada unidad de información puede tomar cualquier valor entre 0 y $2^8-1=255$ en base 2. Sin embargo, en las nuevas máquinas de 16 bits (basadas por ejemplo en el 8086), las 16 líneas de señal para datos corriendo en paralelo pueden utilizarse para representar números entre 0 y $2^{16}-1=65.535$. Las grandes computadoras poseen sub-buses de datos de hasta 64 líneas que les permiten transportar valores entre 0 y $2^{64}-1\simeq 1{,}8\times 10^{19}$, ¡de una sola vez! Más adelante en este capítulo veremos cómo números mayores y menores que éstos, así como otros tipos de datos tales como caracteres o números fraccionarios pueden a pesar de todo representarse y procesarse en máquinas pequeñas a base de dividirlos en grupos de 8 ó 16 bits. Estas máquinas más pequeñas pueden representar y procesar números del mismo tamaño que las grandes a costa de utilizar un mayor número de ciclos de máquina para ello y, en consecuencia, operando a velocidades menores que las máquinas mayores. Sin embargo, ¡una máquina pequeña con pocos usuarios puede dejar fuera de juego a una máquina mayor con una multitud de gente trabajando en ella!

La figura 2.3 muestra varios «dispositivos» colgados del bus principal. Entre ellos hay: una fuente de alimentación, la Unidad

Central de Proceso, memoria y varios controladores. A continuación echaremos un vistazo a estos dispositivos, comenzando por la fuente de alimentación.

Fuente de alimentación

La tarea de la fuente de alimentación consiste en convertir la corriente eléctrica que toma del exterior (normalmente corriente alterna de 120 voltios y 60 Hz.) en el tipo de señal que necesita el bus de alimentación; a saber, corriente de 8 voltios, con una ligera ondulación si la regulación final de potencia se hace en las placas. Para el bus S-100 del antiguo modelo 8080, se necesita además un suministro de, aproximadamente, 20 voltios, también con una ligera ondulación.

La unidad central de proceso

La Unidad Central de Proceso constituye el cerebro o centro de control de la computadora. Mirando globalmente cualquier computadora moderna, la CPU está formada por una o varias placas acopladas al bus principal. Sin embargo, en muchos casos, la CPU ocupa en realidad sólo una pequeña parte de la circuitería de la placa. En cualquier caso, la CPU puede verse como un sistema que «cuelga» del bus principal, alimentado por el sub-bus de alimentación, capaz de enviar y recibir señales de control del sub-bus de control, capaz de enviar direcciones al sub-bus de direcciones y de transferir datos desde y hacia el sub-bus de datos. En mini y maxi-computadoras, la circuitería correspondiente a la CPU suele estar constituida por una matriz poco densa (en el sentido lógico) de elementos electrónicos, normalmente montados sobre chips diferentes. En una microcomputadora, por el contrario, la CPU está formada normalmente por unos pocos chips (1, 2 o tal vez 3) de una alta densidad de componentes (LSI y/o VLSI[1]) constituyendo lo que se llama el *microprocesador* junto con hasta 40 o más chips de menor densidad (MSI y SSI) con misiones de soporte. Notemos que, aunque frecuentemente llamaremos CPU a los chips microprocesadores, la CPU real incluye algunos elementos más.

Algunas veces la CPU consta de varios microprocesadores, formando lo que se llama un *grupo;* e incluso algunas veces una computadora está formada por varios de estos grupos. Investigaremos estas situaciones con más detalle cuando estudiemos los conceptos de procesamiento (tratamiento) en paralelo y multiprocesos en los capítulos 4 y 5.

---

[1] Normalmente se mide la densidad de un circuito integrado por el número de puertas que contiene. Las puertas son las menores unidades lógicas de un circuito, y realizan funciones simples como la AND, OR o NOT. Un dispositivo que contenga menos de una docena de puertas se dice que tiene una pequeña escala de integración (SSI: Small Scale Integration); de 12 a 100 puertas (los límites no son estrictos) se considera una escala de integración media (MSI: Medium Scale Integration); de 100 a 1.000 se dice que tiene una escala de integración alta (LSI: Large Scale Integration), y de 1.000 puertas en adelante el circuito tiene una escala de integración muy alta (VLSI: Very Large Scale Integration). Los chips 8086 y 8088 contienen varios miles de puertas, considerándose por tanto dispositivos VLSI.

La memoria

La memoria de la computadora es otro dispositivo que se encuentra conectado al bus principal, distribuido en una o más placas del sistema. Su misión consiste en almacenar, de una forma básicamente temporal, datos y programas. La memoria puede verse como una colección de celdas individuales, cada una de las cuales lleva asociado un número al que se le da el nombre de dirección. Una celda dada no almacena su dirección; muy al contrario, la CPU y la circuitería de control se sirven de la dirección para localizar y seleccionar una celda particular en la memoria. Todas las transferencias de información de celda a celda o entre celdas y la CPU se hacen vía el bus de datos, utilizando el bus de direcciones para seleccionar las celdas y el bus de control para iniciar y realizar el proceso.

La mayoría de los microprocesadores permiten transferir datos de una celda a otra (transferencias de memoria-a-memoria; en contraposición a las transferencias memoria-a-CPU o CPU-a-memoria), sólo bajo una modalidad especial llamada «de acceso directo a memoria» (DMA: Direct Memory Access). En esta modalidad, la CPU cede el control del bus principal a un dispositivo que recibe el nombre de Controlador de DMA. En el capítulo 6 estudiaremos una versión uni-chip de este controlador.

En la actualidad, prácticamente todas las memorias son memorias a semiconductores (circuitos a semiconductores integrados y encapsulados en un chip). Las memorias más antiguas se construían a partir de pequeños anillos de material magnético entrelazados por delgados hilos conductores. Estas memorias (recibían el nombre de memorias de ferritas o memorias de anillos magnéticos) dejaron de fabricarse debido a su alto coste y gran tamaño.

Desde el punto de vista lógico, existen dos tipos de memorias muy utilizados en los microprocesadores: las RAM y las ROM. En las memorias RAM (Random Access Memory: Memoria de acceso aleatorio) se puede acceder directamente a cualquier celda de la memoria especificando su dirección. Tanto las RAM como ROM son memorias de acceso aleatorio. La diferencia entre ambas reside en el hecho de que, mientras que en una RAM se puede *leer* y *escribir* (almacenar y extraer información), en las ROM sólo se puede leer (Read Only Memory: Memoria de sólo lectura). La información se almacena en las ROM durante un proceso especial llamado de *programación* o *reprogramación*. Durante el proceso de funcionamiento normal, cualquier intento de introducir información que se realice no afecta a la ROM en absoluto.

Los nombres dados a las memorias quizás no sean muy acertados, puesto que desde un punto de vista técnico ambas memorias RAM y ROM son RAM en el sentido que permiten un acceso aleatorio (Random Access). Para distinguirlas, sería más conveniente llamar RWM (Read/Write Memory: Memoria de lectura/escritura) a aquellas que permiten introducir y extraer información, conservar el nombre de ROM para las ya citadas de sólo lectura, y llamar «SAM» (Sequential Access Memory:

	RWM	ROM
RAM	Acceso aleatorio Lectura/escritura 1	Acceso aleatorio Sólo lectura 2
SAM	Acceso secuencial Lectura/escritura 3	Acceso secuencial Sólo lectura 4

Ejemplos:

1. Memoria principal que se puede modificar directamente.
2. Memoria principal que no se puede modificar directamente.
3. Cinta magnética.
4. Cinta de papel (sólo puede perforarse una vez).

Tabla 2.1: Tipos de memoria.

Memoria de acceso secuencial) a aquellas memorias externas tales como cintas magnéticas, en las que para acceder a una posición determinada es necesario acceder antes a todas las posiciones anteriores a ésta (ver tabla 2.1). A pesar de estas deficiencias de nomenclatura, y debido a lo extendido de su uso, seguiremos llamando memorias RAM y ROM a las de lectura/escritura y de sólo lectura respectivamente.

Incluso en las máquinas de 16 bits la memoria suele estar organizada en celdas de memoria de 8 bits (1 byte=1 octeto), debido a que, a pesar de ser máquinas de 16 bits, deben poder trabajar con datos de 8 bits. Tanto los procesadores de 16 bits, como incluso los de 32 bits tienen instrucciones que permiten trabajar con datos de distintos tamaños, entre ellos el octeto de 8 bits. Por ejemplo, para introducir y editar programas, resulta vital el poder procesar tipos de datos como los caracteres provenientes del teclado, que tienen un tamaño de 1 octeto. En una máquina de 16 bits, los octetos se agrupan de dos en dos formando palabras de 16 bits. Los dos octetos de la palabra tienen direcciones consecutivas. Por dirección del par (o dirección de la palabra) entendemos la posición de memoria en que comienza éste; es decir, la posición del primer octeto de la pareja que forma la palabra. Comenzando por la dirección cero, se divide toda la memoria en palabras. Los primeros octetos de cada palabra ocupan las direcciones pares; los segundos octetos las posiciones impares (véase la figura 2.4), por lo que todas las palabras poseen direcciones pares. En algunas máquinas como el Motorola MC68000, o el LSI-11 y PDP-11 de Digital Equipment Corporation; esta es la única manera de acceder a una cantidad de 16 bits. En particular, si en el PDP-11 se intenta acceder a una palabra de 16 bits con una dirección impar, el sistema operativo responde con un mensaje de error.



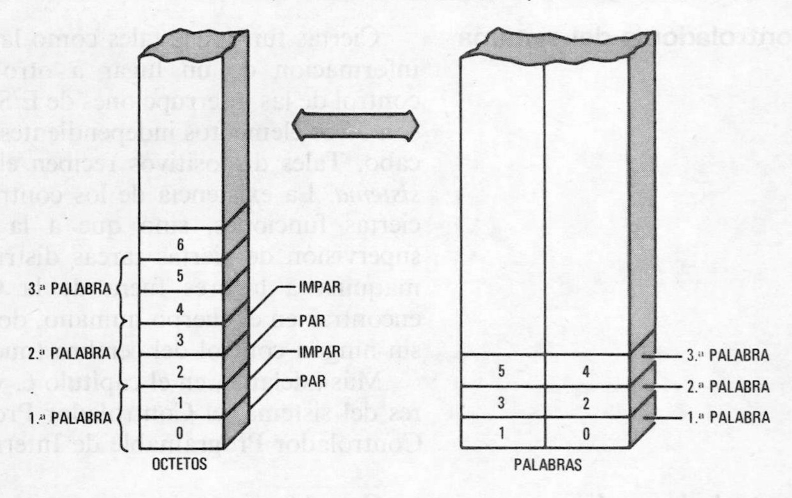


Figura 2.4: Dos aspectos de la misma memoria: desde el punto de vista de los octetos y de las palabras.

En contraste con los procesadores de Motorola y Zilog, el procesador de 16 bits 8086 de Intel no le impone esta restricción al programador. El programador puede definir y acceder a palabras en posiciones pares o impares, eso sí, a un cierto precio: el acceso a una palabra de dirección impar en el 8086 es más lento que el acceso a una palabra de dirección par.

Este retraso se debe a que, para acceder a una palabra de dirección impar, el 8086 debe hacer dos búsquedas: una a la palabra que contiene el octeto impar (y un octeto par inútil) y otra a la palabra que contiene el octeto par (y un octeto impar inútil). El procesador yuxtapone ambos octetos para formar la palabra, y desecha los dos restantes (véase la figura 2.5).

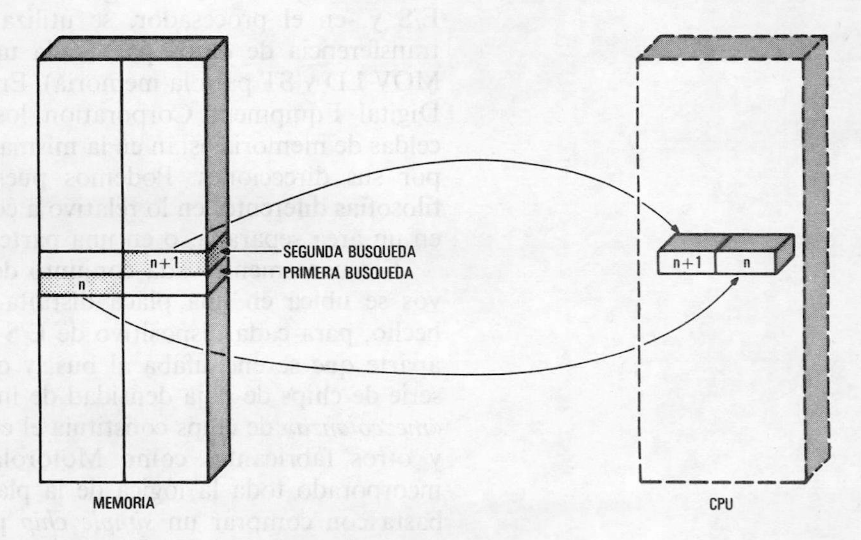


Figura 2.5: Acceso a una palabra impar.

Controladores del sistema

Ciertas funciones tales como las transferencias de bloques de información de un lugar a otro (transferencias DMA), o el control de las interrupciones de E/S se realizan más eficazmente si son otros elementos independientes de la CPU los que los llevan a cabo. Tales dispositivos reciben el nombre de *controladores del sistema*. La existencia de los controladores no sólo hace posible ciertas funciones, sino que a la vez libera a la CPU de la supervisión de ciertas tareas distribuyendo la inteligencia de la máquina a lugares fuera de la CPU. Algo similar podemos encontrar en el cuerpo humano, donde ¡los actos reflejos ocurren sin ningún control del cerebro (nuestra CPU)!

Más adelante, en el capítulo 6, veremos dos chips controladores del sistema, el Controlador Programable de DMA 8237 y el Controlador Programable de Interrupciones 8259.

Controladores de dispositivos

Cuando vimos el sistema pre-computador completo dijimos que había un cierto número de dispositivos conectados a la computadora central, como eran los teclados, monitores de videos, discos flexibles o impresoras. Tales elementos reciben el nombre de dispositivos de entrada/salida (E/S). Un dispositivo de E/S nunca se conecta directamente al bus principal de la computadora, sino que a través de una especie de cable, se conecta a su propio (o compartido) *controlador del dispositivo*, que es el que a su vez se conecta al bus principal. Los controladores de dispositivo actúan de *interfaz* entre los dispositivos y la computadora mientras que los *controladores del sistema* realizan funciones internas.

A cada controlador del sistema y de dispositivo se le asocia una dirección de la misma forma que se hizo con las celdas de memoria. En los chips preprocesadores de Intel y Zilog, las direcciones asignadas a los dispositivos de E/S se mantienen en un *área* separada del área de memoria. Las señales de control del bus de control permiten distinguir entre los accesos a memoria o a E/S y, en el procesador, se utilizan distintas instrucciones de transferencia de datos para cada una (IN y OUT para E/S, y MOV LD y ST para la memoria). En el Motorola y el PDP-11 de Digital Equipment Corporation los dispositivos de E/S y las celdas de memoria están en la misma área y son distinguibles sólo por sus direcciones. Podemos pues observar que existen dos filosofías diferentes en lo relativo a colocar las direcciones de E/S: en un área separada, o en una parte del espacio de memoria.

Frecuentemente cada conjunto de controladores de dispositivos se ubica en una placa distinta. Hace unos pocos años de hecho, para cada dispositivo de E/S se debía comprar una placa aparte que se enchufaba al bus, y que estaba formada por una serie de chips de baja densidad de integración (SSI y MSI). Esta *«mezcolanza»* de chips constituía el controlador del sistema. Intel y otros fabricantes como Motorola y Texas Instrument han incorporado toda la lógica de la placa en un chip, y así, ahora basta con comprar un *simple chip* para tener, por ejemplo, un controlador de discos flexibles. Se han desarrollado muchos otros

chips controladores de dispositivos, algunos de los cuales son de *propósito especial,* como el controlador de discos flexibles, o el de CRT; y otros son de *propósito general* como el controlador de interfaz paralelo, o el de serie. Estos dos últimos pueden utilizarse para buscar gran variedad de dispositivos de computadora. Los controladores del sistema tales como el controlador de DMA, o el controlador de interrupciones también se han integrado en un solo chip. Actualmente existen chips como el Receptor-Transmisor Asíncrono Universal Multifunción 8256, que realizan muchas funciones, incluyendo el control del sistema.

El proceso de miniaturización ha causado un profundo efecto en la fabricación de las placas, al haber simplificado en gran medida su diseño y al permitir diseñar placas que pueden realizar gran variedad de funciones, incluso funciones del sistema o de E/S. Los más recientes diseños incluyen conexiones para varias interfases serie y paralelo, así como temporarizadores y controladores de interrupción.

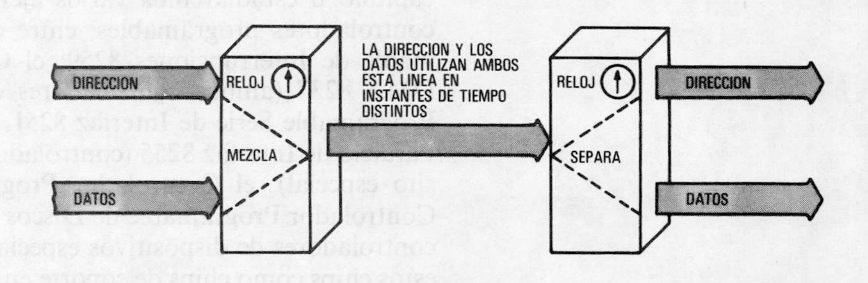
Con la miniaturización se ha conseguido también una mayor flexibilidad y sofisticación. Los controladores del sistema y de los dispositivos son ahora *programables,* de manera que pueden adaptarse y realizar varias funciones bajo control software. En el capítulo 6 estudiaremos varios ejemplos de diferentes tipos de controladores programables, entre ellos el Controlador Programable de Interrupciones 8259; el Controlador Programable de DMA 8237 (ambos controladores del sistema), el Controlador Programable Serie de Interfaz 8251, el Controlador Programable Paralelo de Interfaz 8255 (controladores de dispositivos de propósito especial), el Controlador Programable de CRT 8275 y el Controlador Programable de Discos Flexibles 8272 (estos últimos controladores de dispositivos especializados). Se utilizan también estos chips como chips de soporte en nuestro modelo de computadora, más adelante en este mismo capítulo.

Diseño integrado y el uso de lógica intermedia

La línea de productos integrados Intel es tan amplia que, al menos en teoría, es perfectamente posible construir una computadora completa a partir de ellos, y precisamente esto es lo que haremos en nuestros ejemplos. Sin embargo, la práctica más común es seleccionar unos cuantos chips microprocesadores LSI y VLSI, junto con aquellos chips de funciones especiales que mejor se adapten a las necesidades del sistema, y completar el diseño con chips de menor complejidad encargados de «adaptar» las señales provenientes de chips de familias distintas. La circuitería lógica de adaptación hace como de «pegamento» a la hora de reunir los diferentes chips en una única computadora. En contraste con los componentes mayores, que son chips LSI o VLSI, construidos típicamente con tecnología Metal-Oxido-Semiconductor (MOS), o alguna pequeña variación de ésta, los chips que forman este «pegamento» suelen estar construidos con tecnología bipolar en lo que se llama lógica transistor-transistor (TTL: Transistor-Transistor Logic), con baja o media escala de integración (SSI o MSI). (Para una explicación de las diferencias entre

las tecnologías MOS y bipolar, ver Meindl, James D: «Microelectronic Circuit Element», *Scientific American,* septiembre, 1977). Texas Instrument es uno de los fabricantes más conocidos de dispositivos TTL, y posee uno de los catálogos más completos de chips SSI y MSI. Tanto es así, que aunque otros fabricantes ofrecen estos circuitos TTL, el esquema de numeración de Texas se ha convertido en un esquema estándar en la industria de circuitos integrados.

Gracias a esta lógica intermedia («pegamento») de Texas, y también, aunque en menor escala de Intel, la máquina puede codificar y decodificar las señales de control y separar y combinar señales de direcciones y de datos provenientes de los chips procesadores y controladores. En los sistemas actuales se hace imprescindible el poder combinar estas señales debido a que, en general, hay más señales necesarias que líneas para transportarlas. El chip microprocesador 8086, por ejemplo, tiene sólo 40 terminales y produce muchas más señales lógicas distintas. Este problema aparentemente imposible de resolver se soluciona mediante un *multiplexado en el tiempo* y una *codificación* de las señales (véase la figura que sigue).



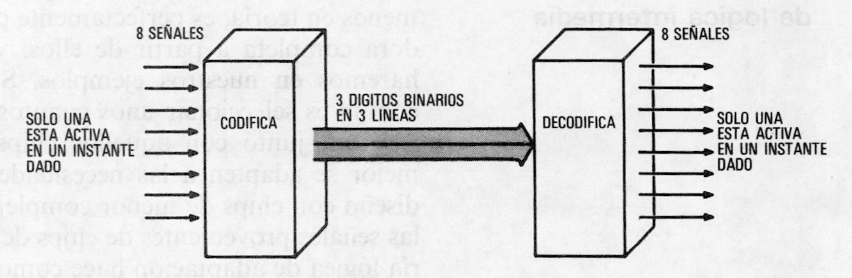


Figura 2.6: Multiplexado y codificación.

Multiplexado en eltiempo

El *multiplexado en el tiempo* se basa en que no todas las señales deben enviarse simultáneamente. Un mismo conjunto de líneas (conectadas a los terminales de los chips) se utiliza para

enviar conjuntos distintos de señales en instantes de tiempo diferentes. Por ejemplo, los procesadores 8085, 8086, 8087, 8088 y 8089 utilizan los mismos terminales para las direcciones que para los datos (en las máquinas de 8 bits, los 8 bits de datos comparten terminales con los 8 bits menos significativos de la dirección). Durante un acceso a memoria, la dirección se envía en el primer ciclo de reloj y los datos en ciclos de reloj posteriores. Para que la lógica intermedia pueda separar externamente estos dos tipos de señal con más facilidad, existe una señal, que recibe el nombre de ALE (Adress Latch Enable), que indica cuando la información enviada es una dirección. Gracias a ella, parte de la circuitería intermedia externa a la CPU puede trabajar con las direcciones e ignorar los datos, mientras que otra parte de esta misma circuitería puede ignorar las direcciones y guardar información sobre los datos.

Codificación

La *codificación* se basa en el hecho de que hay ciertas combinaciones de señales que nunca se van a dar. Con este sistema, varias señales se agrupan y *codifican* en números antes de ser enviadas; se envían y se *decodifican* antes de su utilización. Por ejemplo, el 8086 puede utilizar el bus del sistema de ocho maneras distintas. Cada manera corresponde a un «estado» diferente del procesador (y, en consecuencia, del sistema). A cada forma de utilización del bus se le puede asignar un número de 0 a 7 expresado en binario, de manera que basta con tres líneas para enviar información sobre el estado del procesador. En este caso, son suficientes tres terminales para enviar ocho señales distintas.

De cara al control, es preferible siempre tener señales separadas para cada estado. Cada una de las líneas de control están activas durante un cierto tiempo correspondiente al instante en que el procesador está en ese estado particular. En la práctica, algunos estados no requieren ninguna línea de control, otros sólo una, y otros incluso dos. Así, por ejemplo, durante el estado de *lectura en memoria,* en el microsegundo correcto el procesador debe activar una línea de control que indique a la memoria que debe enviar datos. En uno de los modos de operación del 8086 se generan dos señales de lectura; una de preparación, para avisar a las memorias (más lentas que el procesador) que se va a requerir información de ellas, y otra para indicar cuándo se deben liberar realmente los datos. En total, el 8086 necesita alrededor de siete señales de control. En uno de sus dos modos de operación (el modo mínimo), el 8086 produce un conjunto mínimo de señales de control, y en el otro modo (el modo máximo) envía al Controlador de Bus 8288 un número binario de tres dígitos que representa su estado codificado. El Controlador de Bus 8288 a partir de este mensaje y de las señales de reloj genera el conjunto completo de señales de control. De esta manera, Intel ahorra cuatro terminales del procesador 8086 a costa de obligar al diseñador a utilizar otro dispositivo, el Controlador de Bus 8288,

que se encarga de la decodificación de las señales. En este sentido, el Controlador de Bus 8288 es un ejemplo de lógica intermedia de Intel.

Intel ha desarrollado su propio bus, el Multibús, con un conjunto de placas que los popularizan, «integrando» de esta manera un nivel más alto que el de chip. En particular, Intel ofrece toda una línea de computadoras uni-placa (iSBX) que utilizan los chips 8080/8085/8086 y 8088 como CPU, una amplia gama de placas de memoria; varias placas de E/S y de propósito general, y varias placas controladoras de discos flexibles. Ofrece también chasis para estos elementos, incluyendo un chasis industrial resistente, y los Sistemas de Desarrollo Intelect. En la mayoría de las computadoras uni-placa basados en el 8086 y 8088 (los iSBX 86/12A e iSBX 88/40) existe un módulo especial (iSBX 337) que reemplaza la CPU por un *grupo* que incluye una CPU 8086/8088 asociada al NDP 8087.

El Multibús es más conocido en la industria que en el mercado de las computadoras personales o en el mercado profesional. Estas placas se utilizan mucho en control industrial y en dispositivos de monitorización. Todos estos sistemas los construyen compañías ajenas a Intel a las que se les da el nombre de Fabricantes de Equipo Original (OEM: Original Equipment Manufacturers), que incorporan placas Intel, a veces incluso como computadoras autónomas. Otras compañías como AMD y NEC están construyendo también placas para el Multibús.

Frecuentemente, los chips procesadores y controladores de Intel se utilizan en la construcción de circuitos digitales que deben adaptarse a diseños y normativas estándar distintas a las de Intel. Por ejemplo, los chips Intel se utilizan para proveer de funciones clave a computadoras que usan el popular S-100. El bus S-100 tiene distintas necesidades de control, temporización y multiplexado (así como de alimentación), diferentes a las de los más modernos microprocesadores 8086, 8088, 8089 y 8087, debido a que está basado en la antigua CPU 8080. Para adaptar estos sistemas se necesitan bastantes chips SSI a modo de lógica intermedia.

Hoy en día, en un sistema completo hay normalmente más chips SSI y MSI que LSI o VLSI (aunque actualmente vaya cambiando —véase por ejemplo el Commodore VIC y la Atari 800—), de manera que la lógica se concentra en unos pocos chips (LSI y VLSI), mientras que el resto de chips, aunque necesarios, consumen la mayor parte de la potencia y ocupan la mayor parte de la superficie de la o las placas. La tendencia futura es el tener sólo unos pocos chips muy potentes, interconectados de forma simple y elegante. Intel va tras ello, ofreciendo chips cada vez más potentes.

SISTEMAS BASADOS EN EL 8086/8088

La figura que sigue muestra un diagrama de bloques de un sistema formado por una serie de chips Intel interconectados, formando una computadora. Casi todos los chips se irán viendo en capítulos posteriores.

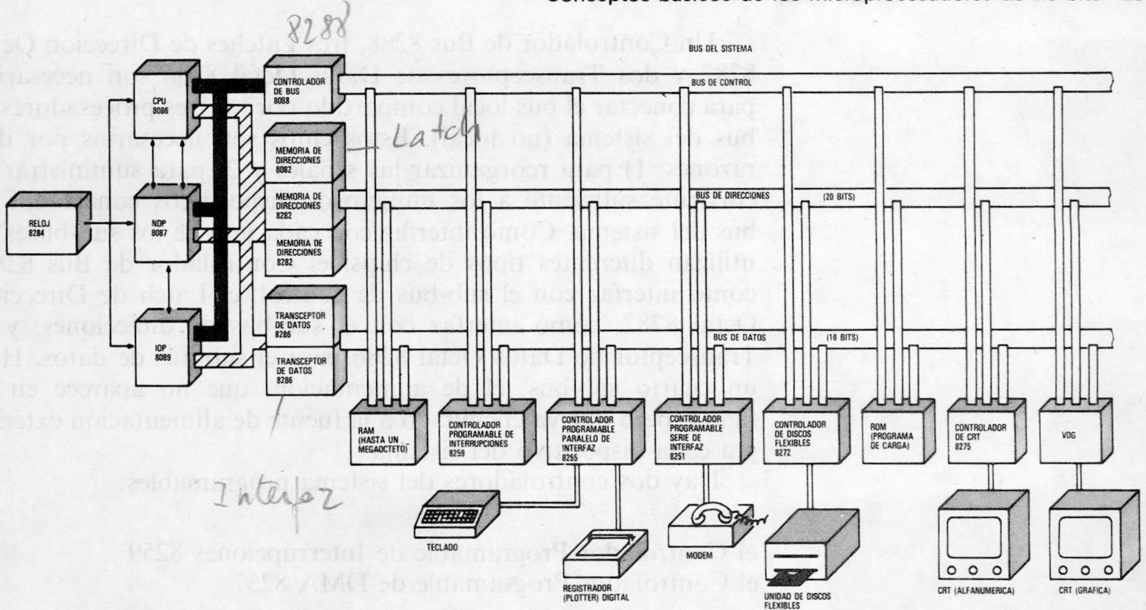


Figura 2.7: Diagrama de bloques de una computadora de 16 bits basado en el procesador de 16 bits.

En particular, el diagrama muestra cómo se conectan los tres procesadores de Intel siguientes:

la Unidad Central de Proceso (CPU) 8086
el Procesador de Datos Numérico (NDP) 8087
el Procesador de Entrada/Salida (IOP) 8089

Estos tres procesadores forman una potente y completa unidad de tratamiento de datos, localizada en la parte izquierda de la figura. La CPU 8086 se encarga de gestionar todo el sistema, distribuyendo tareas a los otros procesadores. Los cálculos aritméticos de alta precisión se realizan en el Procesador de Datos Numérico 8087, y el Procesador de Entrada/Salida (IOP, Input-/Output Processor) 8089 es el encargado de los movimientos de bloques dentro de la memoria de las unidades CRT y entre la memoria central de la computadora y las memorias del sistema de visualización video.

Este grupo tiene su propia estructura de bus *local*. Las líneas de control, provenientes de la CPU 8086, entran al NDP 8087 y al IOP 8089, y se encargan de decidir quién tiene control del bus local en cada momento (sólo un módulo puede utilizar el bus a la vez).

Hay tres chips de interfaz de bus:

el Controlador de Bus 8288
el Latch de Dirección Octal 8282
el Transceptor de Datos Octal 8286

Un Controlador de Bus 8288, tres Latches de Dirección Octal 8282 y dos Transceptores de Datos Octal 8286 son necesarios para conectar el bus local compartido por los tres procesadores al bus del sistema (no local). Estos chips son necesarios por dos razones: 1) para reorganizar las señales y 2) para suministrar la corriente suficiente a los numerosos dispositivos conectados al bus del sistema. Como interfaz con cada uno de los sub-buses se utilizan diferentes tipos de chips: el Controlador de Bus 8288, como interfaz con el sub-bus de control; el Latch de Dirección Octal 8282, como interfaz con el sub-bus de direcciones, y el Transceptor de Datos Octal 8286 para el sub-bus de datos. Hay un cuarto sub-bus, el de alimentación, que no aparece en la figura, pero que va enchufado a la fuente de alimentación externa y a cada dispositivo del sistema.

Hay dos controladores del sistema programables:

el Controlador Programable de Interrupciones 8259
el Controlador Programable de DMA 8237

Estos dos controladores realizan las funciones internas de transferencias por DMA y control de interrupciones ya discutidas.

Hay también cuatro controladores de dispositivos programables:

el Controlador Programable Serie de Interfaz 8251
el Controlador Programable Paralelo de Interfaz 8255
el Controlador Programable de CRT 8275
el Controlador Programable de Discos Flexibles 8272

Cada uno de los dispositivos externos está conectado a uno de estos controladores, que a su vez se conectan al bus principal del sistema. Nótese que hay dos dispositivos (el plotter digital y el teclado) que comparten los 24 bits del Controlador Programable Paralelo de Interfaz 8255.

Por otra parte, hay un controlador de dispositivos no cubierto, el controlador de pantalla gráfica. Este controlador todavía no está disponible en el catálogo Intel, aunque hay chips similares en el mercado; por ejemplo, el Generador de Visualización Video (Video Display Generator, VDG) Motorola 6847, o el nuevo Controlador de Visualización de Gráficas NEC 7220 (que puede que todavía no haya salido al mercado). El Motorola tiene una resolución media de 192 (vertical) por 256 (horizontal), mientras que el NEC es un dispositivo de altas prestaciones, con una resolución mayor a la que cualquier unidad de visualización común pueda necesitar, y con otras características interesantes, como el llevar incorporados las órdenes de dibujo de líneas. Han habido varios sistemas completos controladores de gráficos en el mercado, cuya lógica estaba distribuida sobre una placa en un conjunto de chips SSI y MSI. Fabricantes como Digital Graphic

Systems, Cromemco, Cambridge Development Laboratory y Scion producen estos sistemas uni-, bi- o tri-placa.

Fijémonos que, de hecho, la memoria no es más que otro dispositivo de los procesadores. El sistema descrito puede soportar hasta 1 megaocteto de memoria RAM y ROM. La ROM se utiliza para almacenar un pequeño programa de autocarga (bootstrap) para la inicialización del controlador de discos flexibles. El programa carga el primer sector del disco flexible en memoria. Este primer sector contiene un programa que a su vez carga en memoria el resto del sistema operativo; éste se encarga de inicializar los diversos controladores, activándolos y dejándolos en disposición de recibir órdenes.

Con muy pocas variaciones (de hecho, con sólo alguna simplificación), podemos modificar nuestro diagrama de bloques y reemplazar la CPU 8086 por la CPU 8088. La CPU 8088 tiene un bus de datos externos de 8 bits, por lo que el bus de datos de 16 bits debe sustituirse por otro de 8. Esta modificación simplifica la conexión de algunos controladores de dispositivos de E/S debido a que en su mayor parte utilizan buses de datos de 8 bits. El NDP 8087 y el IOP 8089 pueden trabajar indistintamente con el bus de datos de 8 bits del 8088 o el de 16 del 8086. Aunque el sistema de 8 bits sea más simple y barato, el de 16 ofrece más prestaciones, sobre todo debido a que el movimiento de datos es dos veces más rápido con el bus de datos de 16 bits, siempre que las transferencias se realicen entre componentes de 16 bits: la CPU, la memoria y el sistema de visualización gráfica (que puede operar de ambas maneras). En particular, las instrucciones pueden obtenerse de memoria con mayor velocidad, y la visualización es más rápida en los sistemas de 16 bits.

La computadora diseñada podría ser muy válida como siste-

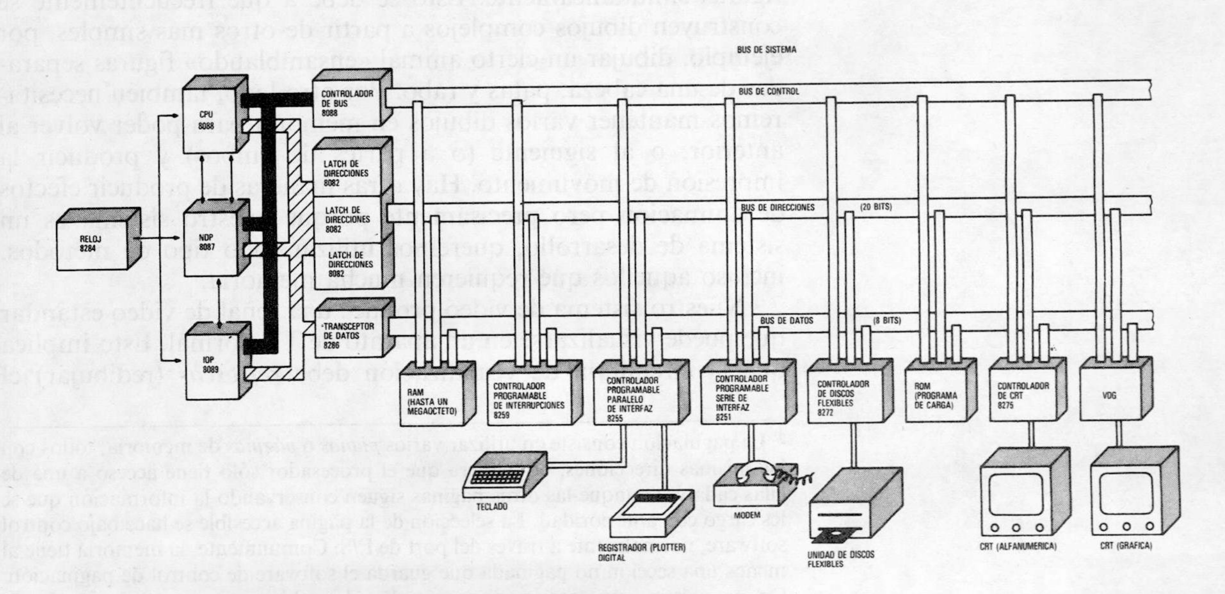


Figura 2.8: Computadora de 8 bits basado en la CPU 8088.

ma de desarrollo de gráficos. El controlador de video visualizaría gráficos de funciones matemáticas más o menos complejos, artísticos, o dibujos animados de escenas complicadas. El Procesador de Datos Numérico 8087 y el Procesador de E/S 8089 serían de gran valor a la hora de manipular eficientemente dichas figuras, mientras que el plotter digital permitiría dibujar las gráficas con una alta resolución. El teclado se utilizaría para entrar y modificar programas. El disco flexible sería útil a la hora de almacenar tales programas (o figuras) y el modem conectaría el computador con alguna red de computadoras, lo que permitiría tener acceso a grandes bases de datos y a ciertos programas públicos. El controlador de CRT se encargaría de visualizar el texto de los programas y cualquier información alfanumérica que se utilizase. Los mayores requerimientos de la visualización de datos serían de sistema. Vamos a estudiar estos requerimientos en mayor profundidad. Supondremos que tenemos un sistema de visualización gráfica ficticio, de las siguientes características: cada dibujo ocupará 96 K octetos. Supondremos tres colores básicos diferentes, con una resolución de 480 puntos verticales por 512 horizontales. Esta es una buena resolución, bastante estándar. Hay por tanto $480 \times 512 = 245.760$ puntos (pixels) en toda la pantalla. Cada pixel necesita de 3 bits (1 por cada color) para guardarse, y por tanto en total se necesitan $245.760 \times 3 = 737.280$ bits. Puesto que cada octeto tiene 8 bits, esto nos da los $92.160 \simeq 93$ K octetos necesarios por dibujo, más de los 64 K octetos que el 8080 o el 8085 pueden direccionar directamente. De hecho, incluso el 8086/8088, con sus 20 bits de direccionamiento, ¡sólo puede almacenar en memoria principal alrededor de 11 figuras a la vez (salvo paginación[2])! Si queremos producir animación, necesitaremos guardar en memoria cuatro o cinco figuras simultáneamente. Esto se debe a que frecuentemente se construyen dibujos complejos a partir de otros más simples, por ejemplo, dibujar un cierto animal «ensamblando» figuras separadas de una cabeza, patas y rabo. Por otro lado, también necesitaremos mantener varios dibujos en memoria para poder volver al anterior, o al siguiente (o a partes de ambos) y producir la impresión de movimiento. Hay otras maneras de producir efectos de animación pero precisamente porque nuestro sistema es un sistema de desarrollo, queremos utilizar todo tipo de métodos, incluso aquellos que requieren mucha memoria.

Nuestro sistema de video produce una señal de video estándar que puede visualizarse en un aparato de TV normal. Esto implica que la circuitería de visualización debe *refrescar* (redibujar) el

---

[2] La paginación consiste en utilizar varios *planos* o *páginas* de memoria, todos con las mismas direcciones, de manera que el procesador sólo tiene acceso a una de ellas cada vez aunque las otras páginas siguen conservando la información que se les cargó con anterioridad. La selección de la página accesible se hace bajo control software, normalmente a través del port de E/S. Comúnmente, la memoria tiene al menos una sección no paginada que guarda el software de control de paginación. De no existir esta sección no paginada, el problema se resuelve guardando duplicados de las partes cruciales del software de control en cada página.

dibujo completo 30 veces por segundo. En realidad, cada imagen (dibujo completo) se divide en dos *campos entrelazados*, cada uno de los cuales se visualiza en 1/60 de segundo. «Entrelazados» significa que cada campo contiene una línea sí y otra no del dibujo, de manera complementaria (entre los dos campos se obtiene el dibujo completo). Este sistema representa un compromiso que por un lado aprovecha los 60 ciclos de frecuencia de la corriente ordinaria, y por otro reduce el *parpadeo* de la pantalla. Una frecuencia de 30 imágenes por segundo es suficiente para eliminar el parpadeo en la mayoría de los televisores normales. Si se desean dibujos de alto contraste se necesita una frecuencia mayor, de unas 40 visualizaciones por segundo. Esto es lo que la red de TV estándar debería permitir. Desgraciadamente, la red de TV se desarrolló mucho antes que los sistemas de visualización gráfica.

Para conseguir animación se necesita cambiar escenas a una razón de 10 imágenes por segundo. No debe confundirse la velocidad de animación con el tiempo de refresco. Así el hardware del sistema visualiza varias veces cada nueva imagen antes de que la computadora la cambie, lo cual es bastante provechoso, porque para la mayoría de las computadoras es difícil re-calcular una imagen nueva 30 ó 40 veces por segundo (la razón de refresco), pero les es sencillo hacerlo 10 veces por segundo (para conseguir una «leve» animación). El proceso entero se realiza de la siguiente manera: Mientras se está visualizando una escena, siempre hay otra que se está dibujando. Cuando la segunda escena se acaba de dibujar, se visualiza y la primera desaparece de la pantalla y queda preparada para ser utilizada como tercera escena. Cada vez que aparece una nueva escena, debe aparecer «instantáneamente», esto es, debe aparecer totalmente en el lugar de la imagen anterior. Este efecto es difícil (caro) de conseguir si el dibujo se calcula durante su visualización, o si se «lleva» a su sitio moviéndola de una posición en memoria al área de visualización. Ni tan siquiera el IOP 8089 puede transferir 92 K octetos con tanta rapidez. Con el bus de datos de 16 bits y una frecuencia de 5 MHz, el IOP necesita del orden de 0,075 segundos para la transferencia. Para conseguir un cambio de imagen «instantáneo», la transferencia deberá realizarse entre dos refrescos, esto es, durante lo que recibe el nombre de *re-trazado vertical*. En los EE.UU., el tiempo de retrazado vertical es de 0,00083 segundos, un tiempo mucho menor que el de transferencia. Es mejor, por tanto, tener un interruptor hardware que permita seleccionar rápidamente qué parte de memoria se visualiza en cada instante.

Frecuentemente nos interesará mover partes de una escena, por ejemplo, un simple rectángulo o uno de los colores. Si restringimos nuestros cálculos a movimientos de una *parcela* de la imagen (digamos 20×20 pixels), sería necesario mover $20 \times 20 \times 3$ bits, esto es, $20 \times 20 \times 3/8 = 150$ octetos.

Esta cantidad, 150 octetos, puede transferirse en algo así como medio milisegundo, incluso utilizando la CPU de 8 bits 8088. Si utilizamos el IOP 8089 en un sistema de 16 bits (basado en el

8086), podremos mover un cuadrado de 52×52 durante un retrazado vertical, es decir, «instantáneamente».

Algunas veces necesitaremos mover dibujos completos, pero esto ocurrirá cuando desarrollemos dibujos «de fondo» de las escenas, y en estos casos, 0,075 segundos no es demasiado malo.

Hemos visto como colabora el IOP 8089 en el movimiento de partes de un dibujo o de la figura entera. Veamos ahora el papel del NDP 8087.

El NDP 8087 es una valiosa ayuda en el tratamiento de números a la hora de producir nuestros modelos matemáticos de los gráficos. Uno de los puntos claves en la producción de tales modelos es la multiplicación de matrices. En el capítulo 5, dedicado por entero al NDP 8087, estudiaremos el programa de multiplicación de matrices, y veremos por qué resulta esencial.

Muy brevemente, supongamos que queremos visualizar un dibujo tridimensional desde distintos puntos de vista, consiguiendo tal vez efectos de animación y perspectiva. Para mover el dibujo tridimensional, usaremos matrices 4×4 que nos permitirán representar convenientemente rotaciones, translaciones, ampliaciones y transformaciones de perspectivas, a través de lo que recibe el nombre de coordenadas homogéneas. La rutina de multiplicación de matrices que daremos en el capítulo 4 multiplicará un punto del dibujo por una matriz 4×4 en menos de 1 milisegundo (ms). De esta manera se pueden procesar 100 puntos en una décima de segundo, justo la frecuencia que necesitábamos para la animación. Con el antiguo sistema basado en el 8085 podíamos estar contentos si conseguíamos una décima parte de velocidad.

TIPOS DE DATOS Y NUMEROS

Un *tipo de datos* es un formato o esquema de codificación que permite representar datos en la memoria de una computadora. Los actuales microprocesadores de 16 bits utilizan varios formatos estándar para representar números y otros tipos de datos como caracteres alfanuméricos. Una descripción detallada de cada tipo de dato queda fuera del alcance de este libro, pero sí daremos al menos una visión general. Cuando lleguemos al NDP 8087 en el capítulo 4, veremos con más detalle la notación de coma flotante.

En nuestro estudio del S-100 vimos un ejemplo de diseño *ortogonal*. Los conductores del bus corrían perpendicularmente a los conectores y placas. Con el almacenamiento de datos y su representación encontramos una situación similar, aunque a un nivel más abstracto. Esta vez las dos cantidades diferentes «corren» perpendicularmente en la tabla de ubicación (ver la figura 2.9). Estas cantidades ortogonales son *unidades de memoria* (bits, octetos, palabras, etc.) y *tipos de datos* (lógicos, ordinales, enteros, números en coma flotante, etc.). Intuitivamente hablando, las *unidades de memoria* son nuestros conductores (realmente llevan información) y los *tipos de datos* son nuestros conectores (necesitan tener acceso a las diferentes unidades de almacenamiento). Igual que no todas las placas necesitaban de todos los

conductores del S-100, no todos los tipos de datos utilizarán todas las posibles unidades de memoria, como puede verse en la tabla.

Para aquellos lectores que prefieran pensar las cosas en términos matemáticos, la relación entre los tipos de datos y su almacenamiento es una «aplicación» en sí, puesto que a un tipo de datos dado, le «corresponde» (usa) varios tamaños de almacenamiento de datos posibles, y a una unidad de memoria le «corresponde» (es usada por) varios tipos de datos diferentes.

La figura 2.9 muestra las necesidades más comunes de memoria para varios tipos de datos. Después de dicha figura hay una breve explicación de cada unidad y tipo.

	UNIDADES DE MEMORIA							
TIPOS DE DATOS	BIT	CUARTETO	OCTETO	PALABRA	PALABRA DOBLE	PALABRA CUADRUPLE	PALABRA DE 80 BITS	BLOQUE
LOGICO	*		*					
ORDINAL			*	*	*	*		
ENTERO			*	*	*	*		
COMA FLOTANTE					*	*	*	
DIGITO BCD		*						
NUMERO BCD		*	*	*	*	*	*	*
CARACTER			*					
CADENA								*
PUNTERO					*			

Figura 2.9: Tipos de datos y su almacenamiento en memoria.

Unidades de memoria

Comenzaremos con las unidades de memoria más pequeñas, los bits, e iremos aumentando el tamaño hasta llegar al concepto de bloque. La figura 2.10 muestra una representación de las diferentes unidades de memoria.

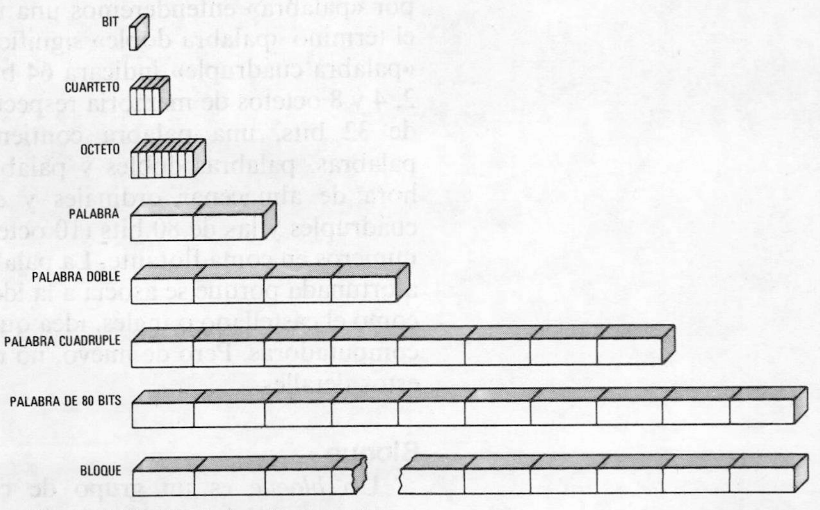


Figura 2.10: Unidades de memoria.

Bit

El *bit* es la unidad más pequeña de información. El término bit proviene de «BInary digiT» (dígito binario). Un bit se almacena y transmite como una señal que puede estar en dos estados; activa (on) o inactiva (off). Puede utilizarse para almacenar variables lógicas (a menudo llamadas indicadores) o números en aritmética módulo 2 (en aritmética módulo 2 sólo hay dos dígitos diferentes, el 0 y el 1); pero también combinado con otros bits, puede almacenar tipos de datos complejos. El nombre de bit es algo desafortunado en este contexto desde el momento que estamos estudiando unidades de memoria y no tipos de datos como el nombre de «binary digit» parece indicar. Sin embargo, aunque lo hayamos mencionado no nos preocuparemos mucho por ello.

Cuarteto

Un *cuarteto* (nibble) son 4 bits, es decir, medio octeto (byte) (como veremos a continuación). Se utiliza fundamentalmente para almacenar dígitos en código decimal-codificado-en-binario (BCD) (números del 0 al 9 codificados en binario) o en hexadecimal (números del 0 al 15).

Octeto

Un *octeto (byte)* son 8 bits, o 2 cuartetos. Puede almacenar un carácter (normalmente codificado en ASCII), un número del 0 al 255, dos números BCD (0 a 99) u ocho indicadores de 1 bit.

Palabra

Contrariamente a la idea más extendida, una *palabra* consta de un número fijo de bits, aunque este número varíe de una computadora a otra. Los microprocesadores de las generaciones actuales tienen palabras de 16 bits, 32 bits, 64 bits, o incluso 80 bits. En este libro que trata de los microprocesadores de 16 bits, por «palabra» entenderemos una unidad de memoria de 16 bits; el término «palabra doble» significará una unidad de 32 bits, y la «palabra cuádruple» indicará 64 bits. Cada una de ellas requiere 2, 4 y 8 octetos de memoria respectivamente. En los procesadores de 32 bits, una palabra contiene 32 bits de memoria. Las palabras, palabras dobles y palabras cuádruples son útiles a la hora de almacenar ordinales y enteros. Las palabras dobles, cuádruples y las de 80 bits (10 octetos) son útiles para almacenar números en coma flotante. La palabra «palabra» tampoco es muy afortunada porque se asocia a la idea de «palabra» de una lengua, como el castellano o inglés, idea que no tiene nada que ver con las computadoras. Pero de nuevo, no nos preocuparemos mucho por estos detalles.

Bloque

Un *bloque* es un grupo de celdas de memoria continuas (octetos o palabras). No tiene tamaño fijo, aunque en ciertos

contextos (como ficheros en disco/disco flexible) un bloque significa un tamaño definido (256, 512 ó 1.024) de octetos. Los bloques se utilizan para almacenar cadenas, trozos de textos o trozos de programas.

Tipos de datos

Veamos ahora *qué* se almacena en estas unidades de memoria, esto es, veamos los datos. Cada *tipo* de dato tiene un cierto formato o codificación que requiere un cierto número de unidades de memoria. Sin una descripción del formato de memoria utilizado, los datos se convierten en algo ilegible y sin sentido, sobre todo cuando utilizamos tipos de datos complicados como puede ser la representación en coma flotante. La figura 2.11 muestra un resumen de cómo se codifican los distintos tipos de datos (de acuerdo con los estándares IEEE, que usan Intel y otras casas). Casi todos los tipos de datos se construyen a partir de «piezas» que utilizan el sistema de numeración binario. Se supone que el lector puede «pensar» en binario, y comprender por qué las computadoras trabajan con este sistema de numeración. En algunas partes del libro utilizaremos también la base 16 (sistema de numeración hexadecimal). Se supone que el lector está también familiarizado con este sistema de numeración. A lo largo del texto distinguiremos los números hexadecimales, anteponiéndoles el carácter *H*. (Si necesita refrescar estos conceptos, vea Waite, Mitchell; Pardee, Michael *Microcomputer Primer* Indianápolis, Indiana: Howard W. Sams & Co., Inc.)

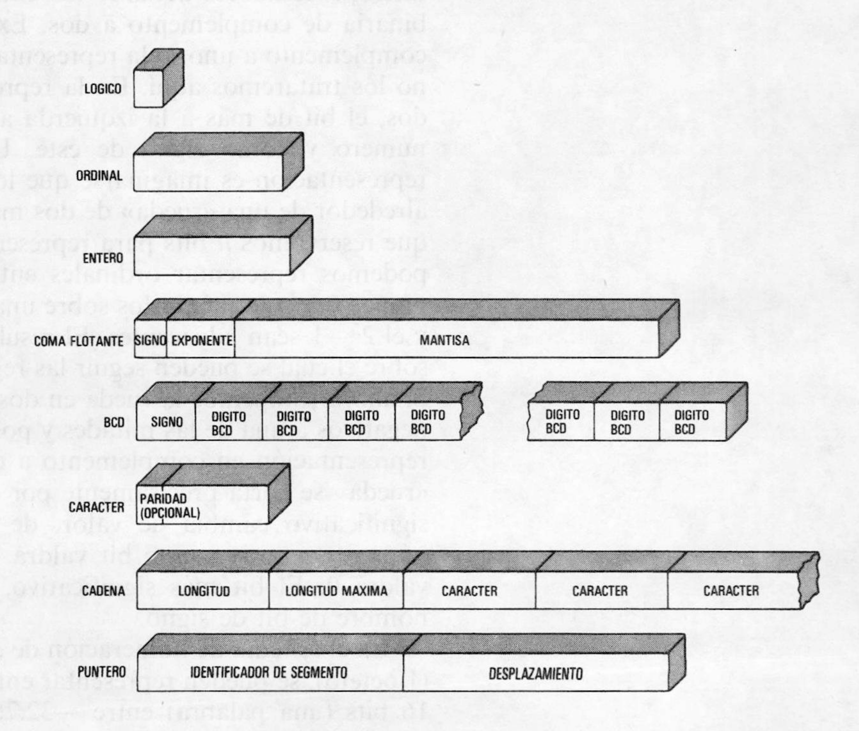


Figura 2.11: Codificación de los distintos tipos de datos.

Comenzaremos por el tipo de datos «lógicos», que requieren un solo bit, y llegaremos hasta los datos de tipo «cadena». No describiremos con demasiado detalle la representación en coma flotante ya que es un tema que se cubrirá en el capítulo 4, dedicado al NDP 8087.

Lógicos

Un dato *lógico* es una cantidad que sólo puede tomar uno de dos valores posibles: verdadero o falso, 0 ó 1, activo o inactivo, sí o no, etc. Se necesita un solo bit para representarlo. Este tipo de dato tiene interés como indicadores condicionales que definan bifurcaciones de programa dependiendo de su valor, o como indicadores de estado que nos permitan conocer si un cierto dispositivo está preparado o no para la entrada o salida.

Ordinales

Son números enteros sin signo, o valores de contadores comenzando en 0. Se almacenan en la computadora en binario, ocupando distintos tamaños (veremos ocho). Si se utilizan ocho bits para guardarlos, el ordinal puede tomar cualquier valor entre 0 y 255; en caso de utilizar 16 bits, el rango del ordinal varía de 0 a 65.535; y con 32 bits o 64 bits, el ordinal puede tomar cualquier valor entre 0 y 4.294.967.296, o $1{,}84 \times E19$, respectivamente.

Enteros

Son números enteros con signo (+ o −). En todos los microprocesadores actuales los enteros se representan en forma binaria de complemento a dos. Existen otros sistemas como el complemento a uno, o la representación en signo magnitud, pero no los trataremos aquí. En la representación de complemento a dos, el bit de más a la izquierda actúa a la vez como parte del número y como signo de éste. Una forma de entender esta representación es imaginarse que los números están distribuidos alrededor de una «rueda» de dos maneras distintas. Supongamos que reservamos n bits para representar cada número. Con n bits podemos representar ordinales entre 0 y $2^n - 1$. Cojamos estos números y coloquémoslos sobre una «rueda», de manera que el 0 y el $2^n - 1$ sean adyacentes. El resultado es un «reloj» gigantesco sobre el cual se pueden seguir las reglas de la aritmética modular. Si ahora rompemos la rueda en dos mitades y asignamos valores negativos a una de las mitades y positivos a la otra, tendremos la representación en complemento a dos (véase la figura 2.12). La «rueda» se corta precisamente por el punto en el que el bit más significativo cambia de valor, de manera que para todos los números negativos este bit valdrá 1, y para todos los positivos valdrá 0. El bit más significativo, con mucha razón, recibe el nombre de bit de signo.

En el sistema de numeración de complemento a dos con 8 bits (1 octeto), se pueden representar enteros entre −128 y +127; con 16 bits (una palabra) entre −32.768 y +32.767, y con 32 bits (doble palabra), el rango de enteros varía entre −2.147.483.648 y

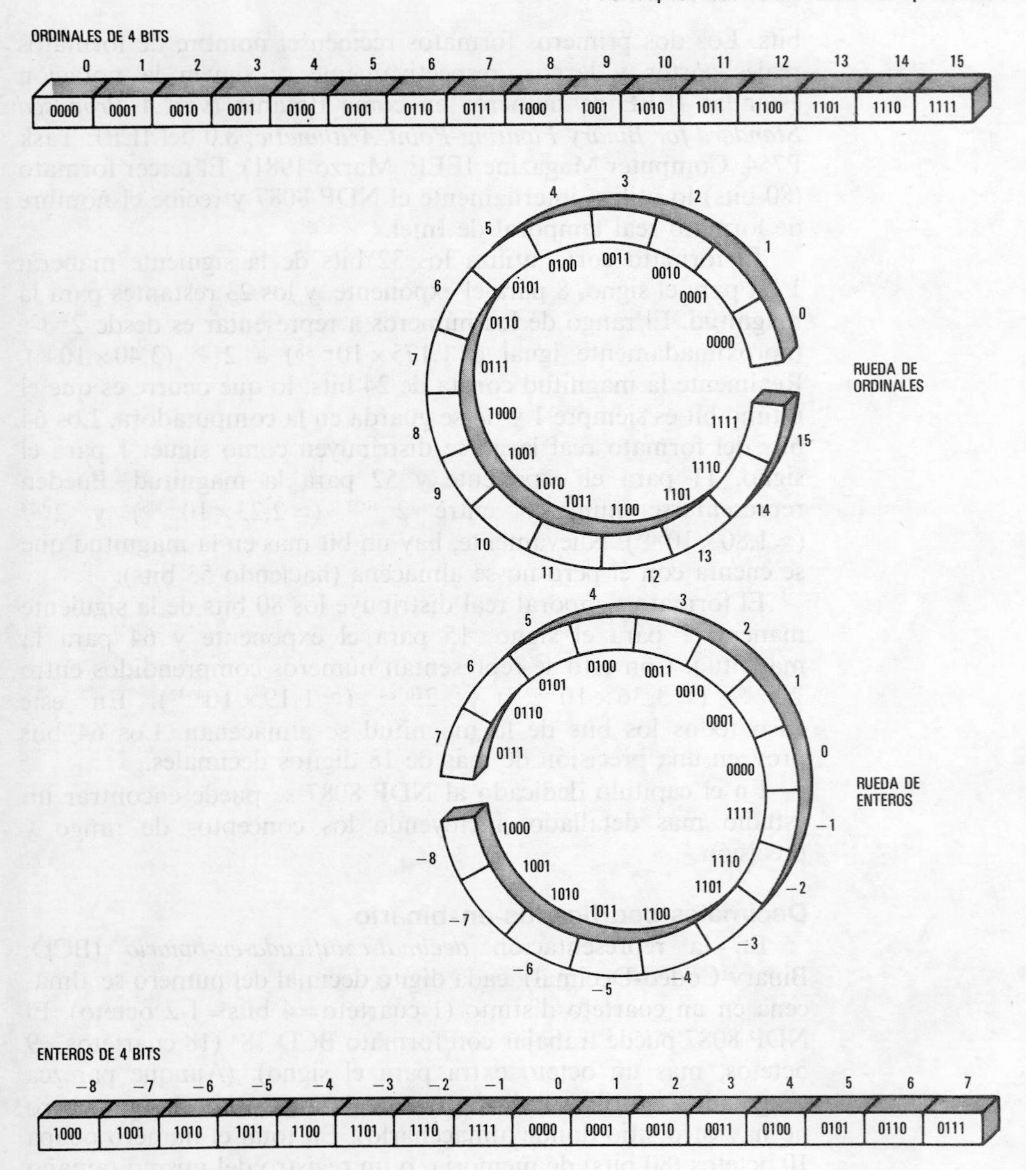


Figura 2.12: Representación de complemento a dos.

+2.147.483.647. Incluso hay enteros que ocupan 64 bits (palabra cuádruple) ¡pudiendo tomar cualquier valor entre (aproximadamente) ±9E18! (esto es, un 9 seguido de 18 ceros).

Coma flotante

La *representacion en coma flotante* está pensada para ofrecer una buena *aproximación* a los números reales. El sistema de representación en coma flotante se parece mucho a la notación científica en la cual cada número viene definido por un signo, una magnitud y un exponente. Se suelen utilizar para su representación 32 bits (palabra doble), 64 (palabra cuádruple) e incluso 80

bits. Los dos primeros formatos reciben el nombre de formatos reales cortos y largos, respectivamente, y siguen la notación estándar IEEE de números en coma flotante (ver *A Proposed Standard for Binary Floating-Point Arithmetic,* 8.0 del IEEE Task P754. Computer Magazine IEEE. Marzo 1981). El tercer formato (80 bits) lo utiliza internamente el NDP 8087 y recibe el nombre de formato real temporal de Intel.

El formato corto utiliza los 32 bits de la siguiente manera: 1 bit para el signo, 8 para el exponente, y los 23 restantes para la magnitud. El rango de los números a representar es desde 2^{-126} (aproximadamente igual a 1.175×10^{-38}) a 2^{128} ($3{,}40 \times 10^{38}$). Realmente la magnitud consta de 24 bits; lo que ocurre es que el último bit es siempre 1 y no se guarda en la computadora. Los 64 bits del formato real largo se distribuyen como sigue: 1 para el signo, 11 para el exponente y 52 para la magnitud. Pueden representarse números entre 2^{-1022} ($\simeq 2{,}23 \times 10^{-308}$) y 2^{1024} ($\simeq 1{,}80 \times 10^{308}$). Nuevamente, hay un bit más en la magnitud que se cuenta con él pero no se almacena (haciendo 53 bits).

El formato temporal real distribuye los 80 bits de la siguiente manera: 1 para el signo, 15 para el exponente y 64 para la magnitud. Con esto se representan números comprendidos entre 2^{-16382} ($\simeq 3{,}36 \times 10^{-4932}$) y 2^{16384} ($\simeq 1{,}19 \times 10^{4932}$). En este caso todos los bits de la magnitud se almacenan. Los 64 bits proveen una precisión de más de 18 dígitos decimales.

En el capítulo dedicado al NDP 8087 se puede encontrar un estudio más detallado, incluyendo los conceptos de rango y precisión.

Decimales-codificados-en-binario

En la representación *decimal-codificada-en-binario* (BCD: Binary-Coded-Decimal), cada dígito decimal del número se almacena en un cuarteto distinto (1 cuarteto=4 bits=1/2 octeto). El NDP 8087 puede trabajar con formato BCD 18^n (18 cuartetos=9 octetos, más un octeto extra para el signo). (Aunque parezca exagerado, se utiliza todo un octeto para guardar el signo, pero no hay otro sitio donde almacenarlo). En total el número ocupa 10 octetos (80 bits) de memoria, o un registro del mismo tamaño en el NDP 8087.

La notación BCD es ideal en aplicaciones de gestión, en las cuales cada centavo en un número de dólares y centavos es importante aunque los cálculos involucren millones o incluso miles de millones de dólares. Las CPU 8086/8088 tienen instrucciones especiales de proceso de números en BCD (DAA: Decimal Adjust for Addition, «Ajuste decimal para suma», y DAS: Decimal Adjust for Substraction, «Ajuste decimal para resta»).

Caracteres

Los *caracteres* se utilizan para representar letras del alfabeto u otros símbolos como dígitos (0-9) o símbolos de puntuación. Lo más común es utilizar el código ASCII (que, dicho sea de paso, codifica los caracteres en 8 bits) para representar los caracteres.

En ASCII, a cada carácter se le asigna un número de manera única. Por ejemplo la A se representa por un 65, la B por 66, y así sucesivamente. El 0 se representa por un 48, el 1 por 49, etc. La mayoría de los manuales de lenguaje ensablador incluyen una tabla ASCII de equivalencias. La mayor parte de las computadoras actuales utilizan una versión restringida de ASCII, necesitando subconjuntos de 7 o incluso de 6 de los 8 bits. Con el subconjunto de 6 bits no pueden representarse minúsculas, y sí con el de 7 bits. De todas maneras estas versiones restringidas guardan los caracteres en 1 octeto de 8 bits. Los bits extras se utilizan para la detección y corrección de errores (ver capítulo 5 para más detalles sobre los códigos detectores de errores), o para indicar el parpadeo de video inverso u otros atributos.

Cadenas

Una *cadena* es una secuencia de caracteres. Se utiliza para guardar textos. Por ejemplo, los mensajes de error son cadenas y algunos programas de textos consideran el documento a editar como una cadena muy larga (de incluso miles de caracteres). Puesto que las cadenas tienen longitudes dinámicamente variables, se incluyen frecuentemente algunos bits extra con información sobre la longitud máxima y la longitud real de la cadena. Por ejemplo, las variables de cadena en algunas implementaciones BASIC tienen una *cabecera* de 3 octetos, en la cual se indica: 1) que el dato es tipo cadena, 2) la longitud máxima de la cadena y 3) la longitud real de la cadena. En otros casos no se guarda la longitud de la cadena, sino que un carácter especial (una especie de indicador de vuelta de carro) indica dónde termina la cadena. Los actuales procesadores de 16 bits no usan formatos especiales para representar cadenas, aunque suelen poseer a menudo instrucciones especiales que permiten tratar, de una manera general, varios formatos posibles. En particular, el IOP 8089 puede acabar una transferencia de cadenas por *contador* o por *carácter de terminación* (o por ambos).

Punteros

Las CPU Intel de 16 bits utilizan los *punteros* para «apuntar» a direcciones físicas en memorias. Se usan junto con la segmentación, concepto que vamos a estudiar a continuación. En los preprocesadores 8086/8088, una dirección física de memoria se guarda como *dos* cantidades de 16 bits. Ambas cantidades se combinan de una forma especial que veremos luego, formando una dirección real de 20 bits. Un puntero es una palabra doble que almacena esas dos cantidades, el *número de segmento* y el *desplazamiento*.

Los microprocesadores Intel utilizan una forma distinta de asignación de memoria a los tipos de datos más largos que Zilog o Motorola. En los Intel, los bits menos significativos se guardan en los octetos de dirección más bajos, mientras que los otros dos los guardan en los octetos de dirección más alta (véase la figura 2.13).

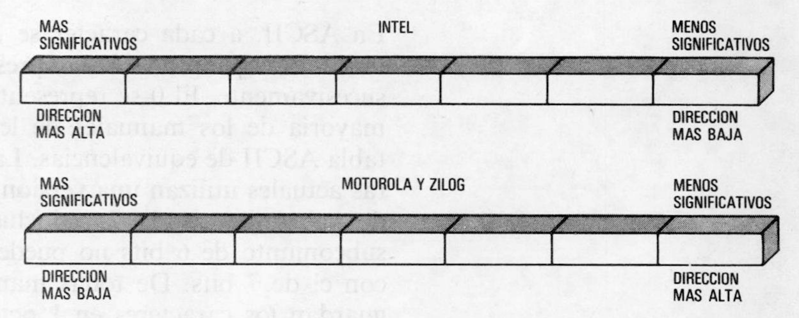


Figura 2.13: Direccionamiento de datos-comunicación entre el sistema utilizado por Intel, Motorola y Zilog.

Estas diferencias se hacen patentes en la representación de números en BCD. En contraposición al sistema de Intel de guardar los dígitos binarios en orden ascendente de derecha a izquierda, el Motorola MC68000 y el Zilog Z8000 guardan los digitos BCD justo en el orden inverso, aumentando de izquierda a derecha.

Otra diferencia es que, en Motorola y Zilog, los datos que ocupan varios octetos deben comenzar en octetos de dirección *par*. Intel permite que comiencen en cualquier octeto aunque, debido a que la CPU 8086 debe acceder a dos palabras para obtener la deseada, el acceso a datos multi-octeto que comienzan en octetos impares resulta algo más lento.

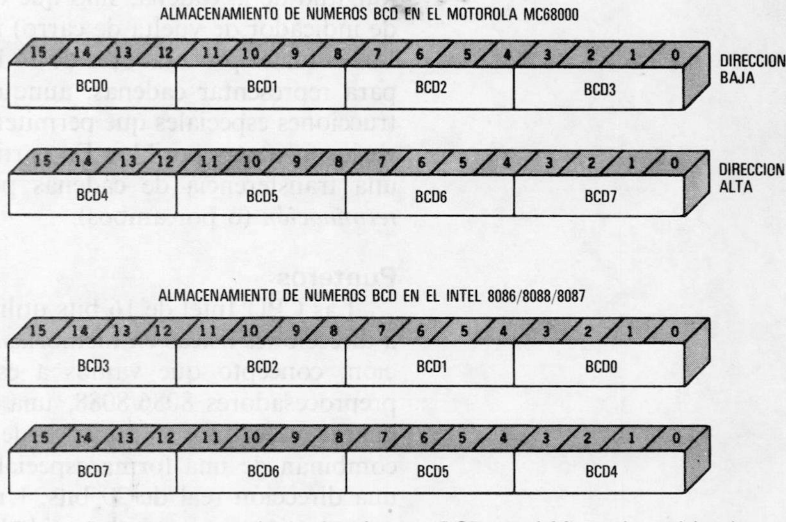


Figura 2.14: Almacenamiento de números BCD en el Motorola y el Intel.

CONCEPTOS BASICOS DE LOS PRE-PROCESADORES DE 16 BITS

En esta sección estudiaremos algunos conceptos fundamentales de los pre-procesadores de 16 bits que, o bien no se daban en los pre-procesadores de 8 bits, o bien tenían un significado distinto. Nos fijaremos en primer lugar en la memoria; la forma como está organizada y la manera en la que se gestiona. Veremos

la organización física y la lógica. Por «física» se entiende la organización de la memoria tal como aparece sobre el bus principal del computador; y por «lógica» nos referiremos a la forma en la que el programador de ensamblador la ve. En los sistemas actuales, más sofisticados, tales distinciones comienzan a hacerse importantes en aplicaciones multi-usuario.

Veremos dos métodos de organización *física:* la *lineal* y la *paginación hardware;* y dos métodos de organización *lógica:* la *segmentación* y la *paginación lógica;* y a continuación estudiaremos cómo, a través de los sistemas de gestión de memoria, se «conectan» las organizaciones física y lógica. Finalmente, veremos un nuevo concepto: cómo un sistema, en particular una computadora, puede tener varios procesadores. Dichos procesadores podrán estar organizados en una relación de igualdad o de maestro-esclavo. Dentro de esta última, describiremos dos sistemas: el *procesamiento en paralelo* y el *multi-proceso,* que aumentan la potencia de los microprocesadores.

Organización física de la memoria

La organización de la memoria en una computadora digital depende no sólo de la computadora particular sino también del punto de vista adoptado (véase la figura 2.15). Comenzaremos con el chip de memoria, para ir subiendo hasta llegar al nivel del sistema (organización física de la memoria). A continuación estudiaremos la organización de la memoria desde el punto de vista del programador (organización lógica), hacia el nivel de sistema (organización física).

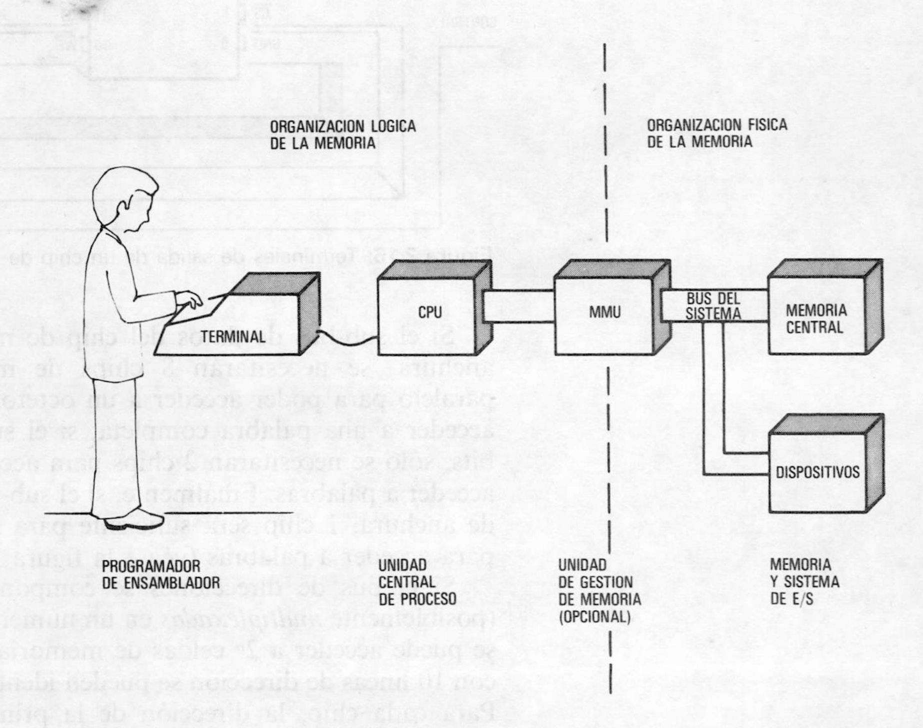


Figura 2.15: Organización física y organización lógica de la memoria.

Gracias a los avances tecnológicos actuales en la integración de circuitos, existen chips de memoria cada vez mayores. Cada año, alguien encuentra la manera de empaquetar más bits en un chip. Las distintas organizaciones de la memoria dentro del chip pueden tal vez entenderse mejor si miramos los terminales de salida de estos chips, y cómo se conectan con el mundo exterior. Para un chip de memoria, las líneas de señal que van a parar a los terminales de salida forman un bus, con los habituales sub-buses de alimentación, control, dirección y datos (véase la figura 2.16). Dependiendo del número de modelo del chip, los sub-buses de datos y direcciones tienen distintos tamaños. El sub-bus de datos puede tener un sólo bit de anchura, 4 u 8 según los casos. El sub-bus de direcciones tiene de 10 a 16 líneas de señal.

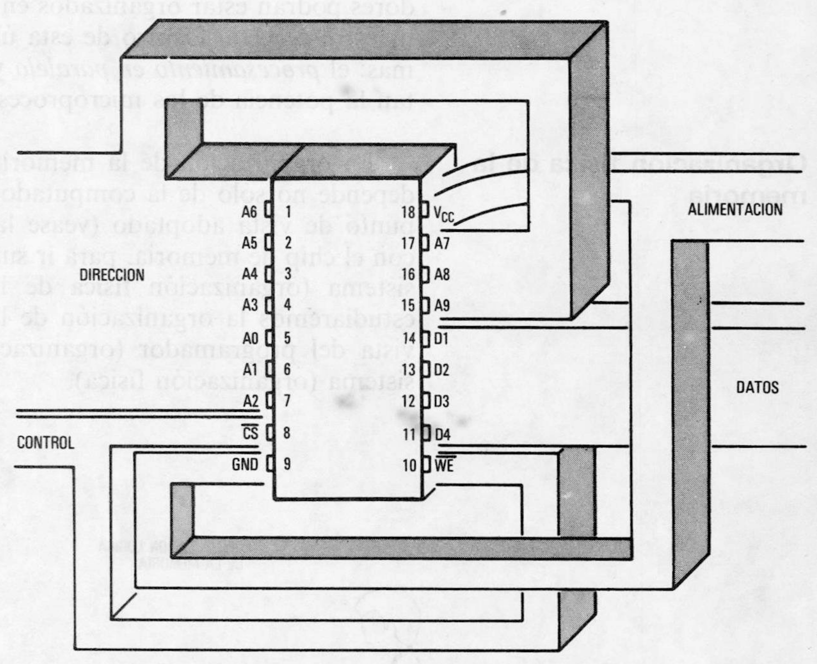


Figura 2.16: Terminales de salida de un chip de memoria típico.

Si el sub-bus de datos del chip de memoria tiene un bit de anchura, se necesitarán 8 chips de memoria conectados en paralelo para poder acceder a un octeto, o 16 chips para poder acceder a una palabra completa; si el sub-bus de datos tiene 4 bits, sólo se necesitarán 2 chips para acceder a octetos, o 4 para acceder a palabras. Finalmente, si el sub-bus de datos tiene 8 bits de anchura, 1 chip será suficiente para acceder a octetos y dos para acceder a palabras (véase la figura 2.17).

Si el bus de direcciones se compone n líneas de señal (posiblemente *multiplexadas* en un número menor de terminales), se puede acceder a 2^n celdas de memoria distintas. Por ejemplo, con 10 líneas de dirección se pueden identificar $2^{10} = 1.024$ celdas. Para cada chip, la dirección de la primera celda es la 0 y la dirección de la última celda es la $2^n - 1$.

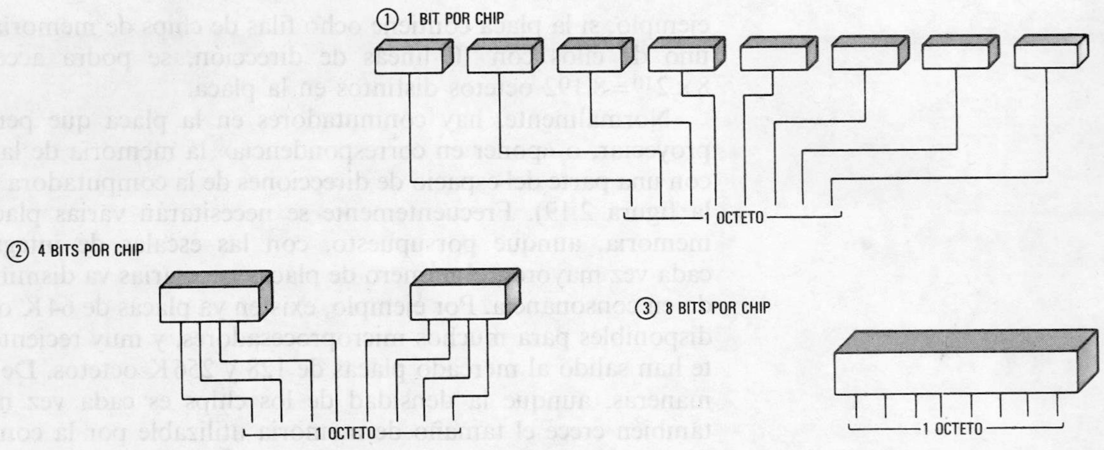


Figura 2.17: Acceso a 1 byte mediante chips de memoria de distintos tamaños (anchura del bus de datos).

Los chips de memoria se conectan en placas para formar memorias mayores. Se suelen colocar formando matrices rectangulares (véase la figura 2.18), de manera que las columnas corresponden a bits diferentes del octeto o palabra, y las filas corresponden a rangos de direccionamiento distintos. Por

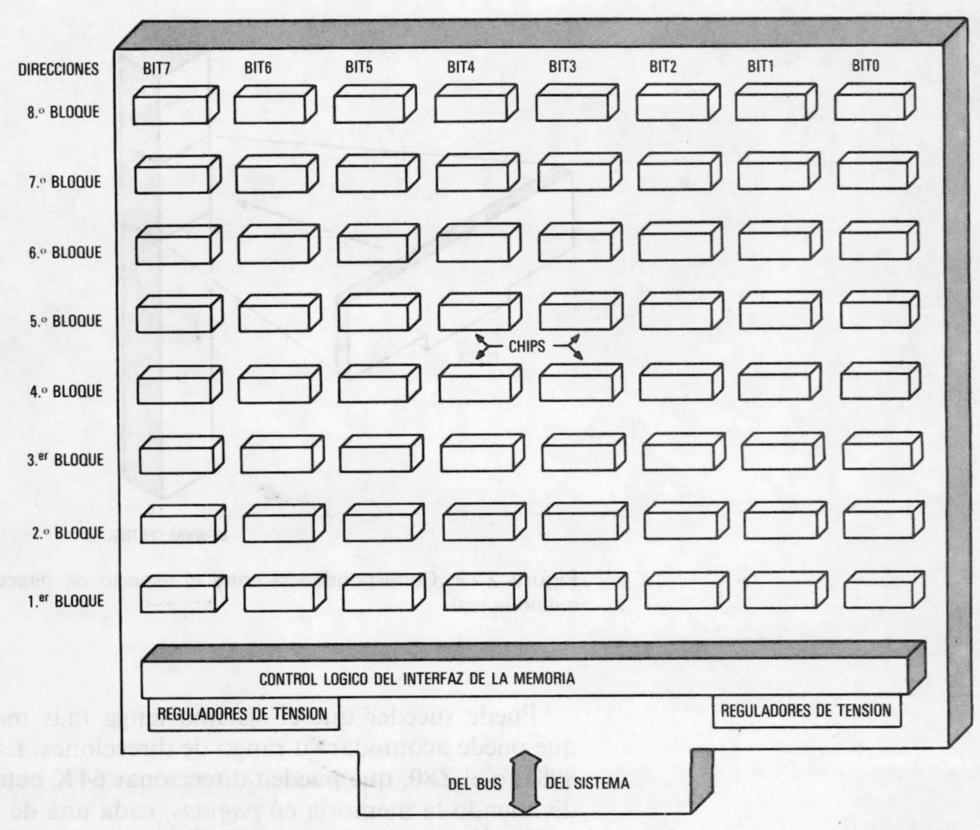


Figura 2.18: Placa de memoria típica.

ejemplo, si la placa contiene ocho filas de chips de memoria cada uno de ellos con 10 líneas de dirección, se podrá acceder a $8 \times 2^{10} = 8.192$ octetos distintos en la placa.

Normalmente, hay conmutadores en la placa que permiten proyectar, o «poner en correspondencia» la memoria de la placa con una parte del espacio de direcciones de la computadora (véase la figura 2.19). Frecuentemente se necesitarán varias placas de memoria, aunque porsupuesto, con las escalas de integración cada vez mayores, el número de placas necesarias va disminuyendo en consonancia. Por ejemplo, existen ya placas de 64 K octetos disponibles para muchos microprocesadores, y muy recientemente han salido al mercado placas de 128 y 256 K octetos. De todas maneras, aunque la densidad de los chips es cada vez mayor, también crece el tamaño de memoria utilizable por la computadora, independientemente de cuanta memoria quepa en cada placa. Nótese que, por otro lado, aunque la memoria se organice como largas líneas, la línea completa no necesariamente debe tener memoria «viva» asignada; esto es, puede haber espacios prohibidos. Si a todas las posibles direcciones no se les asigna memoria real, cuando se intente leer alguna de estas direcciones no asignadas, a la computadora devolverá «todo dígitos 1» (o todo dígitos 0, según el convenio tomado).

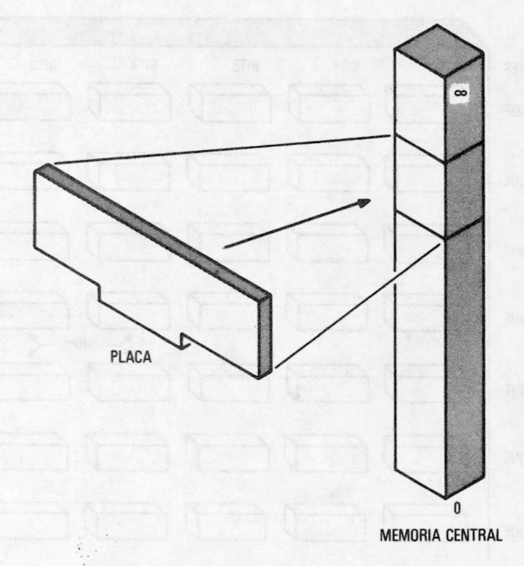


Figura 2.19: Correspondencia entre el espacio de direcciones y la placa de memoria real.

Puede suceder que el sistema tenga más memoria real de la que puede acomodar su rango de direcciones. Es el caso del 8080, 8085, o el Z80, que pueden direccionar 64 K octetos y se consigue dividiendo la memoria en *páginas,* cada una de las cuales sí tiene una correspondencia directa con la memoria real. Con un

conjunto adicional de líneas de direccionamiento se selecciona una de las posibles páginas cada vez. Este esquema de direccionamiento recibe el nombre de *paginación hardware*. Ocurre a veces que el enviar la dirección de la página resulta bastante más lento que enviar una dirección normal. Para obviar este problema, la dirección correspondiente a la página se modifica lo menos posible de modo que el usuario trabaja normalmente en la misma página por largos períodos.

Un posible esquema de paginación frecuentemente utilizado en las computadoras de 8 bits consiste en enviar la información de selección de página a través de un port de E/S a las placas de memoria del sistema. En cada placa, una parte de la lógica se encarga de decodificar dicha información y activar o desactivar los chips de memoria de acuerdo con ella. De esta manera, los procesadores de 8 bits pueden utilizar memorias mucho mayores que los 64 K octetos direccionables.

Organización lógica de la memoria

Acabamos de ver dos formas de organización física de la memoria: 1) como un espacio lineal de direcciones y 2) como un conjunto de páginas cada una de ellas organizada linealmente. Estas son las posibles maneras en las que los otros dispositivos conectados al bus del sistema «ven» a la memoria. A continuación veremos cómo puede organizarse la memoria desde el punto de vista del programador. Llamaremos a ésta *organización lógica* de la memoria.

En el caso más sencillo, la organización lógica de la memoria coincide con la física. Es la situación más frecuente en las microcomputadoras de 8 bits. Sin embargo, las microcomputadoras de 16 bits, con su mayor espacio de direccionamiento, y sus aplicaciones multi-tarea, requieren esquemas de organización más elaborados. Discutiremos dos métodos: la *segmentación* y la *paginación*.

Básicamente, la *segmentación* es un método de acceso a memoria en el cual toda dirección se compone de dos cantidades; un *identificador de segmento* y un *desplazamiento*. El identificador de segmento apunta a un área general de memoria (el segmento), mientras que el desplazamiento apunta a una dirección dentro de ese segmento. En la segmentación, la memoria real está organizada de una forma lineal, en la que cada celda tiene asignada una única dirección que se calcula, por métodos que veremos más adelante, a partir del identificador de segmento y del desplazamiento. El desplazamiento es el valor que aparece en los programas como dirección del dato, o como rótulo, o como dirección de una instrucción del programa; mientras que el identificador de segmento aparece sólo cuando se ha de cambiar de segmento. Es decir, una vez seleccionado el segmento, el programador y el procesador deben recordar en qué segmento se hallan, para poder interpretar las direcciones del programa como direcciones dentro de ese segmento, hasta que se especifique un nuevo segmento. En ese momento se establece el nuevo identificador. Normalmente, los cambios de segmento se hacen al comienzo del programa,

cuando se debe acceder a áreas de datos distintas, o cuando se pasa control de una sección de código a otra.

En las microcomputadoras de 8 bits, programación en lenguaje ensamblador, cada acceso a memoria corresponde a un desplazamiento, y toda la memoria es de hecho, un único y gran segmento. En los procesadores de 16 bits, el espacio de direccionamiento es tan grande que el rango de direcciones accesibles al programador (algunas veces sólo 64 K), suele ser inadecuado para cubrir toda la memoria disponible. La segmentación permite que el programador especifique sus propios rangos dentro del espacio direccionable real. En vez de tener un solo rango, puede ser interesante definir varios rangos de direccionamiento. En los Intel 8086/8088, existen cuatro áreas de este tipo llamadas *segmentos*.

Respecto a la segmentación, hay varias actitudes comunes: 1) no vale la pena, 2) es una buena idea, 3) ¿de quién fue la idea? y 4) ¡no tengo ni idea!

Motorola adopta la primera posición. La segmentación origina gran cantidad de trabajo innecesario y retarda el proceso. En el lado opuesto, Intel y Zilog optan por la segmentación, sobre todo debido a sus ventajas dentro de la multi-programación. Ambas casas tratan de mejorar la programación fomentando la tendencia actual de la modularidad, y mejoran el multi-proceso facilitando en gran medida la escritura de módulos totalmente reubicables. De hecho, cada módulo se puede escribir en lenguaje ensamblador y traducir a lenguaje máquina ¡como si comenzara en la posición 0 de memoria! En *tiempo de ejecución* es cuando se asigna la dirección real de comienzo vía la segmentación hardware.

Las implementaciones Intel y Zilog de la segmentación son bastante distintas. Zilog usa un chip de gestión de memoria (MMU: Memory Managment Unit) adicional que no necesita Intel. Zilog comercializa dos versiones de su microprocesador Z8000: una con segmentación y otra sin ella. La versión con segmentación requiere el chip MMU antes citado para regenerar la dirección a partir de las dos partes que facilita la CPU. Dicho de otra manera, la versión segmentada del Z8000 soporta la división de las direcciones de memoria en dos partes, pero no especifica cómo deben recombinarse para formar de nuevo la dirección real.

Contrariamente, Intel ha incorporado en el propio chip de la CPU un esquema de gestión de memoria muy simple. El chip CPU 8086/8088 tiene cuatro registros especiales llamados *registros de segmentación:* uno para el código (instrucciones), dos para los datos y uno para la pila (almacenamiento temporal especial). Los contenidos de estos registros (los identificadores de segmento) se multiplican por 16 y se suman a la información sobre la dirección proveniente del resto de la CPU (el desplazamiento) para calcular las direcciones de memoria reales (véase la figura 2.20). Con el 8086/8088, los segmentos pueden comenzar en cualquier frontera de 16 bits, y acabar en cualquier posición (hasta 64 K octetos más adelante). Las versiones futuras del

8086/8088 (el iAPX 286) *permitirán* que los segmentos comiencen en *cualquier* posición de memoria. Hay instrucciones especiales que cargan la información sobre los identificadores de segmento en los registros de segmentación. Todo esto lo veremos más detalladamente en el capítulo 3.

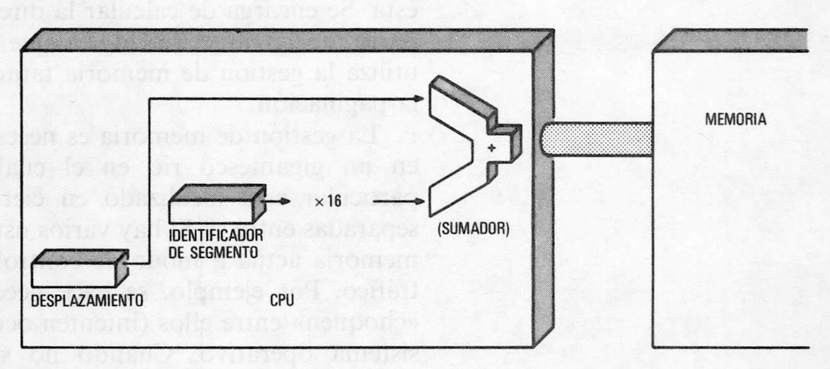


Figura 2.20: Gestión de memoria en la segmentación (Intel).

Hemos visto cómo la segmentación calcula la dirección a partir de dos valores. El otro posible esquema que estudiaremos a continuación es la *paginación*. La paginación también utiliza dos valores para representar una dirección. La memoria se divide en *páginas* lógicas, cada una de las cuales contiene unos cuantos miles de celdas. Cada acceso a memoria se hace a partir de un *identificador de página* que selecciona la página deseada, y un *desplazamiento* que localiza la posición de la celda en la página. La paginación lógica puede realizarse incluso si la memoria está organizada linealmente, pero si la memoria está físicamente organizada en páginas, es muy interesante utilizar las mismas páginas físicas que lógicas para evitar retardos innecesarios que se pueden originar al tener que cambiar de página física varias veces dentro de una misma página lógica.

Aunque parezcan muy similares, hay varias diferencias entre la paginación y la segmentación: 1) las páginas tienen un tamaño fijo, mientras que el tamaño de los segmentos puede variar; 2) la segmentación permite que los segmentos comiencen y acaben en cualquier posición de memoria, mientras que las páginas deben estar comprendidas entre unas fronteras de bloque rígidas; 3) los segmentos se pueden superponer y las páginas no, y 4) la segmentación obliga al programador a estar atento a los límites del segmento, mientras que en la paginación este aspecto suele hacerse automáticamente, sea vía hardware, sea a través del sistema operativo.

En general, la segmentación es un esquema más flexible, y como tal requiere a menudo una mayor atención por parte del programador, que debe recordar que tal trozo de programa o de datos está en el mismo segmento lógico que otro. El sistema Intel requiere que el programador defina realmente el comienzo y final

de segmento. Tal vez esto puede ser una molestia al principio, pero al fin y al cabo, hace que el programador sea consciente de lo que significa en realidad la programación modular.

Gestión de memoria

La gestión de memoria actúa de interfaz entre el esquema lógico de direccionamiento de memoria y la organización física de ésta. Se encarga de calcular la dirección física real de memoria a partir de la dirección lógica que el programador le define. Se utiliza la gestión de memoria tanto en la segmentación como en la paginación.

La gestión de memoria es necesaria si se piensa en ésta como en un gigantesco río en el cual el tráfico para un usuario particular está localizado en ciertas regiones, a menudo muy separadas entre sí. Si hay varios usuarios, el sistema de gestión de memoria actúa a modo de controlador o guardia para todo ese tráfico. Por ejemplo, se hace necesario evitar que los usuarios «choquen» entre ellos (intenten ocupar regiones ajenas), o con el sistema operativo. Cuando no se utilizaban los sistemas de gestión de memoria, se producían cantidad de extrañas caídas del sistema cuando el programador entraba en él. Sucedía que el programador entraba sin darse cuenta en zonas del sistema operativo, ¡destruyendo información de éste! Las computadoras actuales como el PDP-11, e incluso algunos de las más recientes microcomputadoras, van provistas de protecciones para evitar estas calamidades. En los dos apéndices que tratan el iAPX 286 y el iAPX 432 se habla de estos esquemas de protección.

En la última sección de este capítulo veremos cómo ha implementado Intel su sistema de gestión de memoria en el propio chip de la CPU, pero primero estudiaremos cómo se realiza la gestión de memoria por medio de chips especializados (MMU).

Una Unidad de Gestión de Memoria para un sistema de segmentación contiene un conjunto de registros (registros de segmentación), cada uno de los cuales contiene la dirección de comienzo de un segmento (zona de memoria). Cuando la MMU recibe las dos partes de la dirección (identificador de segmento y desplazamiento) de la CPU, utiliza el identificador de segmento para seleccionar uno de los registros de segmentación, y suma el contenido de tal registro con el *desplazamiento* (suministrado por la CPU). El resultado es la dirección real de memoria (véase la figura 2.21). La diferencia respecto al esquema de segmentación del Intel 8086/8088 es que aquél, en vez de multiplicar por 16, busca en una «tabla» (registros de segmentación) las direcciones de comienzo de los segmentos. Una versión futura del 8086 (el iAPX 286) utiliza también este procedimiento de búsqueda en «tablas», de manera que los segmentos pueden comenzar en cualquier posición de memoria.

En concreto, el nuevo procesador de 32 bits Intel iAPX 432 usa un esquema de «doble búsqueda». El identificador de segmento se utiliza para localizar un segundo identificador que a su vez busca la dirección real. Con esto se pretende aumentar la

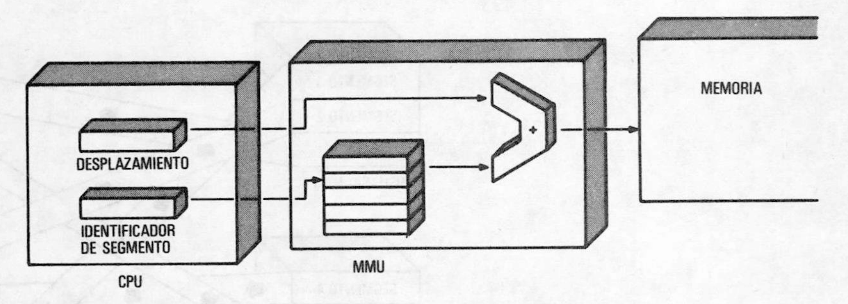


Figura 2.21: Gestión de memoria con segmentación.

seguridad del sistema, detalle importante si se tiene en cuenta que gran parte de las computadoras de hoy en día contienen informaciones personales, militares o financieras que deben mantenerse en secreto, y están conectadas a líneas telefónicas. Si no se controla, prácticamente cualquier persona con una computadora conectada a la misma red podría acceder e incluso modificar estas informaciones privadas. Como protección se utilizan dispositivos hardware y software especializados, como puede ser el sistema de doble búsqueda.

Es importante resaltar que la gestión de memoria requiere un cierto tiempo de proceso. Así, por ejemplo, los registros de segmentación de la MMU deben cargarse previamente con la dirección de memoria apropiada para que la gestión sea correcta.

Cuando se utiliza la gestión de memoria en un sistema de paginación, los registros de la MMU reciben el nombre de registros de paginación y contienen números de páginas. Cuando la MMU recibe las dos partes de la dirección (en este caso el identificador de página y el desplazamiento), utiliza el identificador de página para seleccionar el registro de paginación que contiene el número de página de la celda de memoria buscada. El desplazamiento sirve de nuevo para determinar la posición relativa de la celda de memoria dentro de la página.

En algunos sistemas como el microcomputador PDP-11 de Digital Equipment Corporation, o el Motorola MC68000, el usuario «ve» las direcciones como un solo número en vez de dos. La parte más significativa de la dirección la utiliza la MMU como identificador de página o de segmento, y la parte menos significativa como desplazamiento.

Una gran ventaja de los sistemas de gestión de memoria es que el usuario puede trabajar como si toda la máquina fuese para él/ella, aunque de hecho la esté compartiendo con mucha gente. Para ello es necesario un software especial, el llamado *sistema de tiempo de ejecución*. El sistema de tiempo de ejecución asigna un conjunto de registros de segmentación distinto a cada usuario (en contraposición al esquema Intel, en el cual se asigna un registro de segmentación distinto para cada tipo de acceso: código, datos o pila). Cada usuario utiliza las direcciones relativas (desplazamientos) y los identificadores de segmentos como si no existiesen



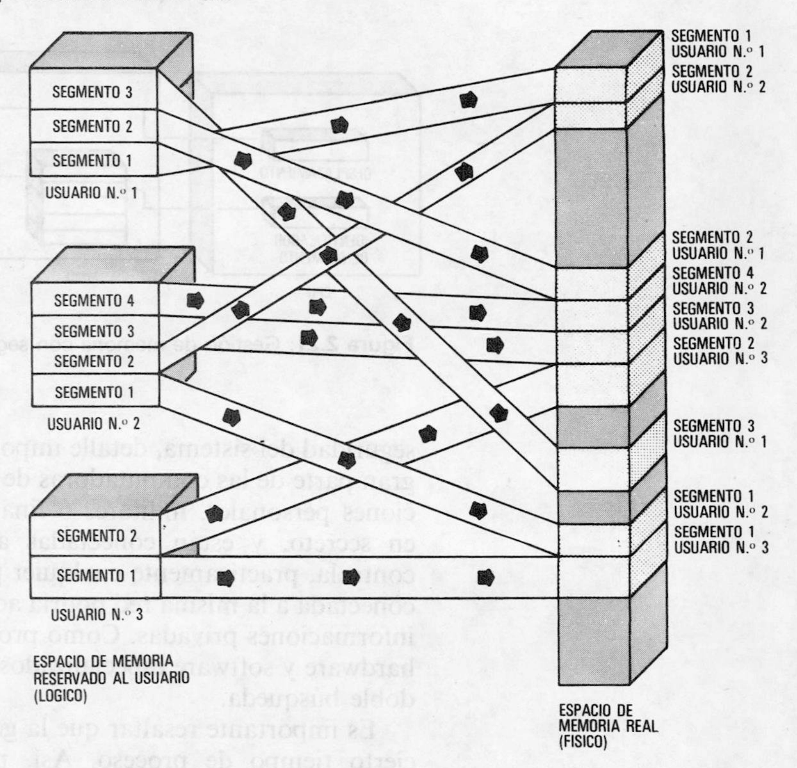


Figura 2.22: Asignación de memoria en multi-proceso.

otros usuarios, sin preocuparse nunca qué segmentos de programas hay realmente en memoria (véase la figura 2.22).

La gestión de memoria es necesaria en los sistemas de multiproceso; esto es, cuando varios programas pueden ejecutarse concurrentemente. Incluso si una sola persona utiliza la computadora, el sistema de gestión de memoria sigue siendo interesante al permitirle ejecutar varios trabajos a la vez; o, si hay un solo trabajo, puede descomponerse en tareas más pequeñas que se ejecuten simultáneamente. El multi-proceso permite utilizar más eficazmente los recursos de la computadora. Así, por ejemplo, los programas y el editor de textos normalmente utilizan recursos de la computadora durante cortos períodos de tiempo. Naturalmente, durante tales intervalos, la potencia total de la computadora (los recursos) es absolutamente necesaria; pero en los espacios de tiempo entre utilizaciones, dichos recursos pueden asignarse a otras tareas. Un buen sistema operativo puede planificar las tareas a realizar, asignándoles y actualizando prioridades, aumentando la eficacia del sistema y agotando las posibilidades de la CPU.

Las microcomputadoras de 8 bits también pueden utilizar un sistema de gestión de memoria, especialmente con esquemas de paginación como los ya expuestos, pero no llevan incorporados métodos para la manipulación de los identificadores de página o de segmento. En estos casos, los identificadores se gestionan vía

software, y se envían a través del sistema operativo y los ports de E/S a las placas de memoria. La gestión de memoria por paginación sigue siendo eficiente en los procesadores de 8 bits, pero la segmentación se complica bastante, hasta el punto de que esta posibilidad suele reservarse a los procesadores de 16 bits, mejor equipados en estos aspectos.

Existen hoy en día sistemas operativos para las microcomputadoras de 8 bits que permiten el multi-proceso. De todas maneras, hay que tener en cuenta que el hecho de soportar varios trabajos simultáneos fuerza de alguna manera las posibilidades y los recursos de la CPU, de forma que tiene más sentido invertir tiempo, dinero y esfuerzos en el desarrollo de técnicas de multiproceso para los sistemas basados en procesadores de 16 bits que en los de 8. Las tendencias actuales indican claramente que el precio del hardware se está reduciendo respecto al coste de trabajo-hombre. Parece, pues, que en un futuro cercano, el hardware será más fácil de diseñar, de utilizar y, sobre todo, más barato globalmente.

Procesamiento en paralelo y multi-proceso

Una importante tendencia actual es la utilización de varios procesadores en una misma computadora. Por ejemplo, la impresora, los terminales o el plotter pueden llevar incorporados sus propios procesadores (incluso a veces la propia computadora lleva un microprocesador menos potente que el de alguno de sus periféricos. Por nombrar un ejemplo de los primeros tiempos de los microprocesadores, el SOL 20, con un 8080, podía utilizar como terminal un Anderson Jacobson equivalente a un IBM Selectric, que llevaba incorporado ¡un Z-80!).

La idea del multi-proceso consiste en incorporar varios procesadores a la computadora para aumentar su potencia de cálculo (dos cerebros mejor que uno). Una posibilidad es tener dos procesadores centrales idénticos en la misma computadora. Es el caso del iAPX 432, 32 bits, que se ha diseñado de forma que puede trabajar en paralelo con ¡256 copias idénticas de sí mismo! El secreto para que todos contribuyan a realizar las tareas eficientemente consiste en dividir los programas en procesos que se ejecuten en procesadores. Esta forma de atacar el problema involucra de alguna manera el concepto de ortogonalidad que ya se discutió a raíz del estudio del bus S-100 y el almacenamiento de tipos de datos, aunque a un nivel más alto. Volviendo al bus, imaginemos que los procesadores juegan el papel de los conductores (conducen procesos), y que cada trabajo es un conector (cada trabajo comprende varios procesos cada uno de los cuales utiliza un procesador distinto). De esta manera, un trozo particular de programa puede ser ejecutado por cualquier procesador. La decisión de qué procesador utilizar para cada proceso depende más de los procesadores que estén libres en ese instante que de qué procesador sea el más adecuado para dicho proceso. Cada vez que se asignan procesadores a los procesos, se dice que se ha formado una correspondencia entre esos dos tipos de objetos, como muestra la figura 2.23.

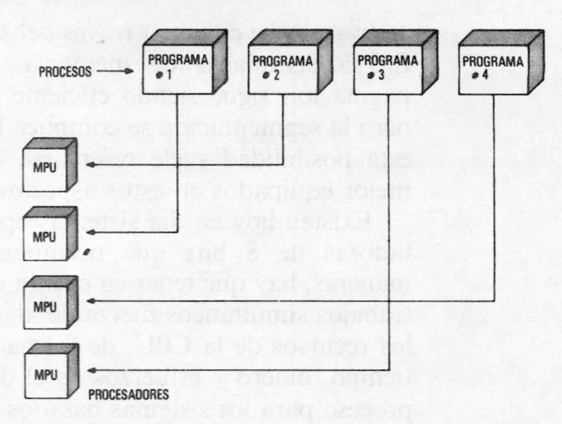


Figura 2.23: Procesos y procesadores.

Otra posibilidad para aumentar la eficiencia del sistema es la de asignar procesadores *esclavos* o *periféricos* bajo el control del procesador central. En este libro veremos dos métodos: el procesamiento paralelo y el multi-proceso. En un sistema de *procesamiento paralelo,* dos o más procesadores comparten el mismo *flujo de instrucciones*. Es decir, todos los procesadores están realizando el mismo programa, turnándose en la ejecución de las instrucciones. La idea principal es que algunas instrucciones se ejecutan mejor en unos procesadores que en otros. Las limitaciones tecnológicas actuales impiden que, por ahora, todos estos microprocesadores se integren en el mismo chip. Como ejemplo de este sistema veremos el Procesador de Datos Numérico NDP 8087 en el capítulo 4.

En un sistema de *multi-proceso,* dos o más procesadores comparten la memoria, pero operan sobre distintos flujos de instrucciones. Un procesador concreto puede tener el control, distribuyendo trabajo entre los otros procesadores a través de mensajes; pero cada uno de ellos, una vez inicializado, ejecuta independientemente del primero su propio programa. Como ejemplo, en el capítulo 5 estudiaremos el Procesador de E/S 8089.

La gran ventaja del procesamiento paralelo y el multi-proceso radica en que el procesador central se ve liberado de una gran cantidad de trabajo, pudiéndose concentrar en la gestión global del sistema. El procesador central se limitaría a distribuir tareas entre otros procesadores especialmente diseñados para realizar estas tareas eficientemente. Por ejemplo, el IOP 8089 gestiona las transferencias de E/S mucho mejor que la CPU, y el NDP 8087 opera con tipos de datos más largos y complicados que la CPU. Todas las aplicaciones se benefician de este sistema más amplio (hay más instrucciones; y, sobre todo, más instrucciones especializadas) y potente (más rápido). Con el IOP 8089 ciertas transferencias se pueden llegar a realizar 16 veces más rápidas que con la CPU; mientras que el NDP 8087 aumenta en un 10 por 100 el rendimiento del sistema!

Por supuesto, para controlar esta cooperación entre procesa-

dores en los sistemas de procesamiento paralelo y multi-proceso, se necesitan algunas instrucciones especiales. Así, por ejemplo, para permitir el procesamiento paralelo, las instrucciones se dividen en conjuntos asignados a cada procesador. Si un procesador recibe una instruccion ajena, no la intenta ejecutar sino que la devuelve al sistema para que la ejecute el procesador correcto. En consecuencia, cada procesador del sistema lleva asociado uno o más conjuntos de instrucciones ficticias. En el capítulo 4 veremos cómo funciona el Co-Procesador de Datos Numérico 8087. La instrucción ficticia del 8086/8088 ESC (ESCape) se utiliza para definir el conjunto de instrucciones del NDP 8087, y el resto de las instrucciones del 8086/8088 son ficticias para el 8087. La situación en realidad es un poco más compleja en el sentido de que el 8086/8088 ayuda al NDP 8087 extrayendo los operandos especificados en la instrucción ESC.

Cabe preguntarse si el procesamiento paralelo es posible con los microprocesadores de 8 bits. Si bien es cierto que algunos procesadores de 8 bits poseen instrucciones ficticias, suelen estar bastante indocumentadas, y no son de fiar en el sentido de que el fabricante puede hacer modificaciones de una serie a otra que afecten al comportamiento de dichas instrucciones. No suele ser recomendable diseñar sistemas de procesamiento paralelo para microprocesadores de 8 bits.

Otra instrucción especial utilizada en sistemas de procesamiento paralelo es la de sincronización. Puesto que todos los procesadores trabajan sobre el mismo flujo de instrucciones, su sincronización es indispensable. Con un sistema 8086/8088/8087, la instrucción WAIT hace que la CPU espere a que el NDP 8087 acabe de ejecutar la instrucción en curso. Para ello se necesita cierto hardware adicional, y una línea desde el NDP 8087 a la CPU 8086/8088 que le indique cuándo ha acabado el primero. La sincronización del procesamiento paralelo puede realizarse totalmente por hardware en vez de la combinación hardware/software propuesta. En ambos casos, los procesadores de 8 bits no están bien equipados para estas tareas.

El multi-proceso requiere instrucciones especiales más «sutiles». Una de estas instrucciones es el LOCK del 8086/8088. LOCK envía una señal al resto de los procesadores del sistema avisándoles que no deben utilizar el bus hasta que se haya completado la instrucción siguiente. La importancia de esta instrucción reside en el hecho de que, en multi-proceso, hay que proteger los datos y programas para evitar que se acceda a ellos simultáneamente desde dos procesadores distintos. Supongamos, por ejemplo, que dos procesadores intentan actualizar el mismo dato a la vez. Cada procesador leerá el dato, lo cargará en uno de sus registros, operará con él y devolverá el valor modificado a memoria. Dependiendo de quién lo haga primero el resultado será distinto. Si el procesador A lee el dato y el procesador B lee la misma posición de memoria antes de que A haya devuelto el valor, ambos procesadores contendrán el mismo valor original del dato. Si suponemos ahora que ambos procesadores operan el dato, y

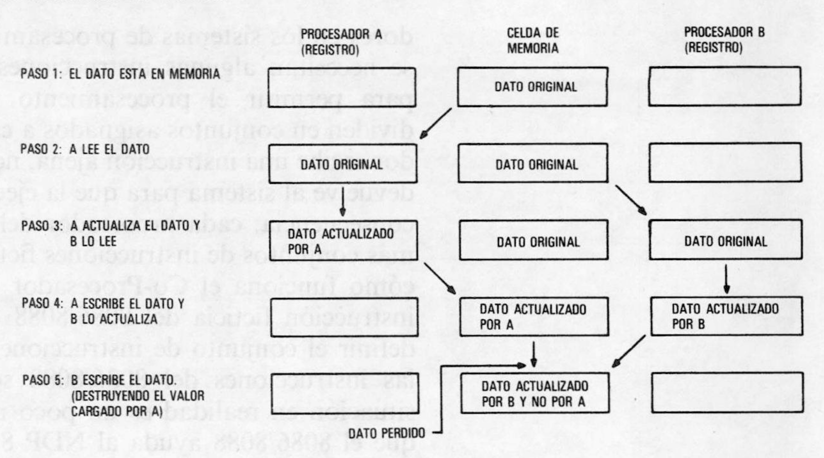


Figura 2.24a: Dato sin protección al que acceden dos procesadores a la vez.

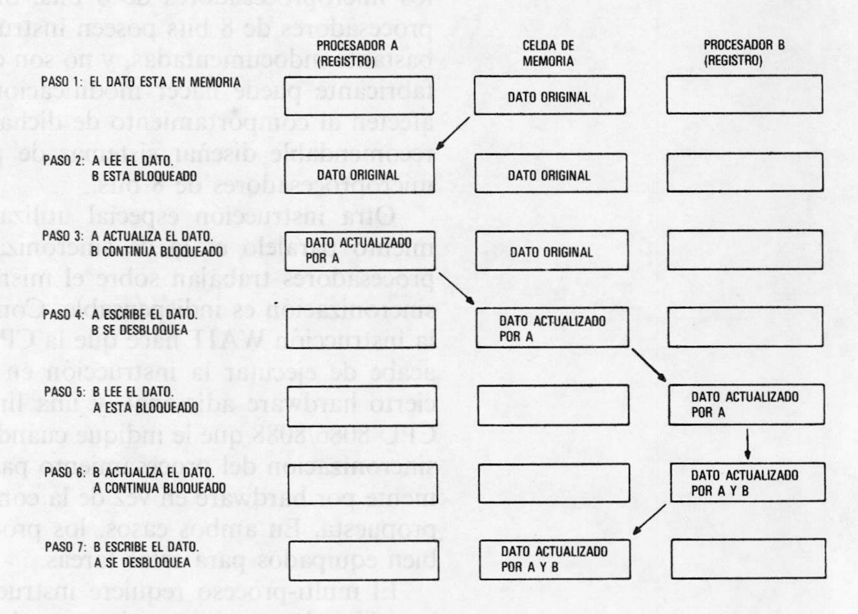


Figura 2.24b: Dato protegido al que acceden dos procesadores.

devuelven el resultado a memoria, primero el A y luego el B, el resultado final es el mismo que si sólo hubiese trabajado el procesador B, ya que al devolver el valor destruye el valor devuelto por A (véase la figura 2.24). El problema de las concurrencias no es sencillo, y la detección de errores en estos sistemas es difícil. Resultados mucho más dramáticos se dan cuando el procesador intenta modificar o cargar un programa en un área de memoria que está siendo utilizada por otro procesador. ¡Pueden suceder cosas imprevisibles!

Para prevenir y evitar conflictos de este tipo entre procesadores, cada bloque de memoria que se utilice para datos o progra-

mas de cualquier procesador del sistema contiene un octeto especial: el octeto de «ocupado». Todo procesador que desee utilizar un bloque de memoria debe preguntar antes por su *octeto de ocupación* para saber si el bloque está o no ocupado. Si no lo está, el procesador activa el octeto de ocupación. El único problema surge cuando dos procesadores preguntan o intentan activar simultáneamente el octeto de ocupación. En tal caso, podría ser que ambos procesadores accediesen al mismo bloque a la vez. La solución a este problema está en utilizar la instrucción LOCK cada vez que se accede al octeto de ocupación. La instrucción LOCK se encarga de evitar que los otros procesadores puedan hacer algo hasta que el procesador en cuestión haya activado el octeto de ocupación, cosa que puede hacerse con la instrucción XCHG (eXCHanGe). Si el bloque está libre, el procesador lo usa; y si el bloque está ocupado, el procesador espera y vuelve a preguntar más tarde (véase la figura 2.25).

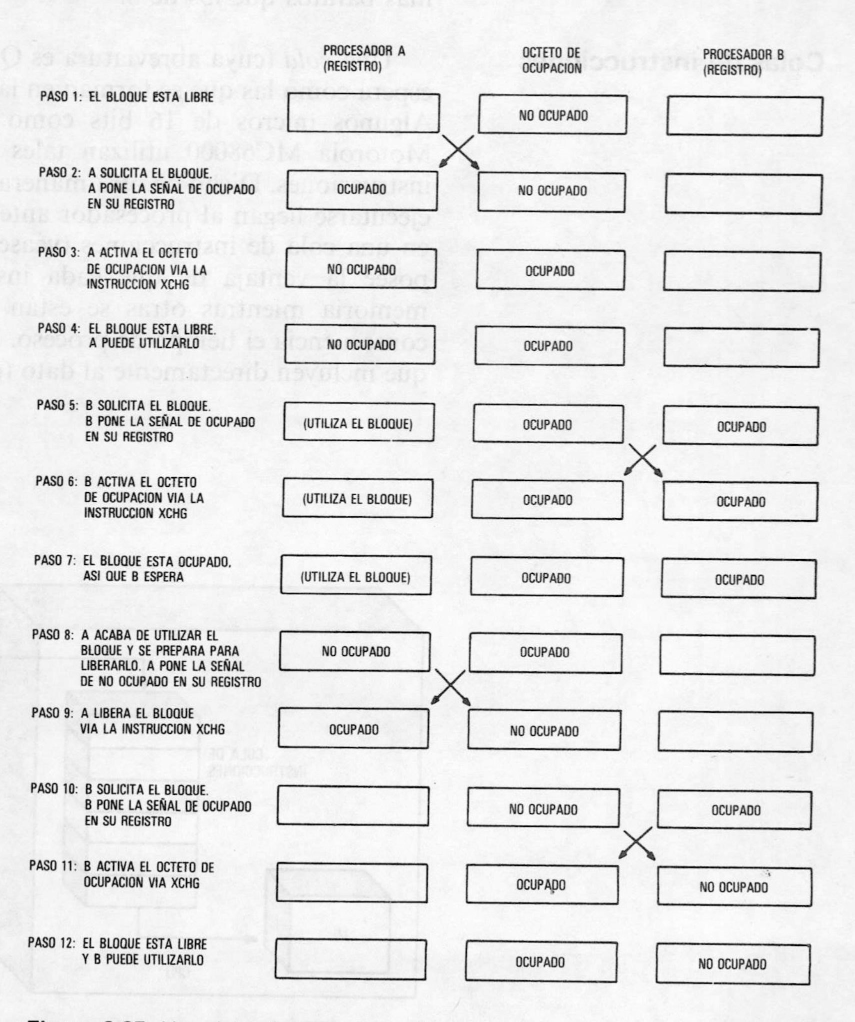


Figura 2.25: Uso del octeto de ocupación.

Como puede verse, este sistema necesita algo de hardware adicional (para las líneas de señal que bloquean a los otros procesadores) y una instrucción especial. Este sistema puede simularse en los procesadores de 8 bits; por ejemplo, la instrucción LOCK puede simularse enviando una señal a un port de E/S especial para que algún circuito específico haga que la memoria no sea accesible temporalmente por los otros procesadores del sistema. El multi-proceso en procesadores de 8 bits es por supuesto posible, pero a cambio de un coste extra. Si el sistema posee memoria dinámica, y necesita por tanto refresco cada cierto tiempo, puede causar serios problemas. Debido a que la lógica de la instrucción LOCK ya viene incorporada en los microprocesadores de 16 bits, el multi-proceso con estos nuevos chips es más fácil y barato. El único costo adicional por la posibilidad de la multi-programación en los procesadores de 16 bits es el de los varios procesadores del sistema; sin embargo, incluso el costo de éstos se está reduciendo de tal manera que dentro de poco serán más baratos que los de 8.

Colas de instrucciones

Una *cola* (cuya abreviatura es Q, de «queue») es una línea de espera como las que se forman en las cajas de los supermercados. Algunos micros de 16 bits como el 8086/8088 de Intel, o el Motorola MC68000 utilizan tales «líneas de espera» para sus instrucciones. Dicho de otra manera, las instrucciones que han de ejecutarse llegan al procesador antes de lo necesario y «esperan» en una cola de instrucciones (véase la figura 2.26). Este sistema posee la ventaja de que cada instrucción puede extraerse de memoria mientras otras se están ejecutando, reduciéndose en consecuencia el tiempo de proceso. Por ejemplo, las instrucciones que incluyen directamente al dato (datos inmediatos), se ejecutan

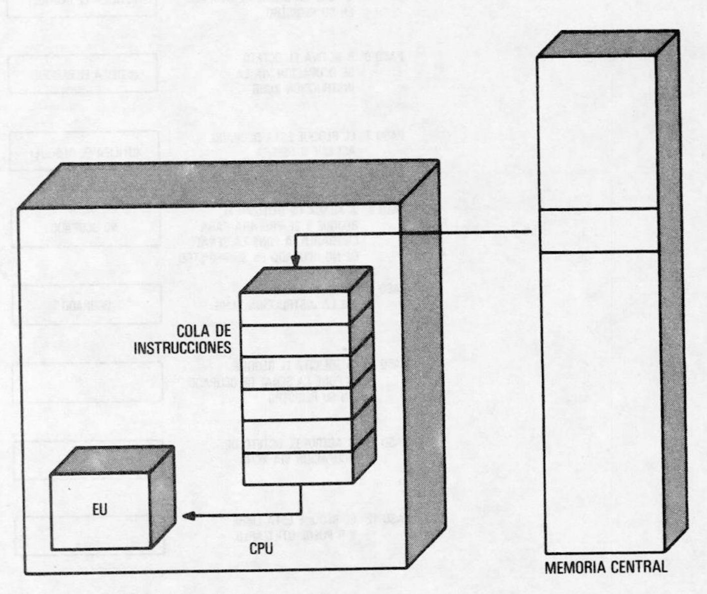


Figura 2.26: Cola de instrucciones.

prácticamente a la misma velocidad que aquellas otras que utilizan datos de los registros de la CPU. Es también muy útil cuando se usa el procesador con memorias lentas tales como EPROM (Erasable Programmable Read Only Memory: Memoria de sólo lectura borrable por programa).

Las colas de instrucciones son normalmente cortas, de 4 a 6 octetos. En concreto la del 8086 es de 6 octetos (tres palabras) y la del 8088 es de 4 octetos. Cada uno de estos dos procesadores está en realidad dividido en dos *subprocesadores* separados, la Unidad de Interfaz con el Bus (BIU: Bus Interface Unit) y la Unidad de Ejecución (EU: Ejecution Unit). La BIU se encarga de controlar las transferencias entre la CPU y el mundo exterior, mientras que la EU es la que realiza las operaciones aritméticas y lógicas. La BIU, entre otras cosas, extrae las instrucciones de la cola conforme se van necesitando, al mismo tiempo que la EU va ejecutando las anteriores instrucciones.

NIVEL DE MICROPROGRAMACION

Normalmente, cuando alguien intenta aprender a programar, comienza con algún lenguaje como el BASIC u otro de los llamados de «alto nivel». Este término significa que el BASIC es un lenguaje de programación «basado» hasta cierto punto en el inglés, que puede comprenderse fácilmente y que no varía demasiado de una computadora a otra, esto es, que es bastante independiente de la estructura de la máquina. Ahora bien, el BASIC o cualquiera de estos lenguajes funcionan a base de llamar a otras instrucciones más primitivas, dependientes de la estructura de la máquina particular.

El principiante debe pues aprender a continuación que existen los llamados lenguajes de bajo nivel. En particular el ensamblador es un lenguaje de bajo nivel que permite un *control directo* de la CPU de la computadora; algo que el BASIC sólo puede hacer en casos muy especiales (con el PEEK, POKE y CALL o USR). Evidentemente cuanto más directo sea el control, más dependerá el lenguaje de la estructura de la CPU. El lenguaje desarrollado para una CPU no sirve para otra. (Todos los lenguajes ensambladores difieren entre sí, pero los BASIC son prácticamente iguales.) Excepto las líneas de comentarios, cada línea de lenguaje ensamblador consta del nombre de una instrucción del procesador (nemotécnico) y de la posición que ocupan los datos de esta instrucción (los operandos). Algunas líneas contienen al principio un rótulo que identifica la línea, y que puede utilizarse para identificar el comienzo de un procedimiento o de un bucle.

El lenguaje ensamblador se ha diseñado para proporcionar una forma fácil de controlar un nivel incluso más bajo de programación, el lenguaje máquina. El lenguaje máquina consta de una secuencia de códigos numéricos almacenados en la memoria del computador. La CPU lee estos códigos y ejecuta las acciones correspondientes. Un programa típico en lenguaje máquina transfiere información octeto a octeto o palabra a palabra de una parte a otra de la máquina, y realiza operaciones aritméticas y lógicas básicas. Cada paso se especifica mediante

una pequeña cadena de datos binarios que se corresponden casi directamente con líneas de ensamblador.

Podría pensarse que el lenguaje máquina es el lenguaje de nivel más bajo que hay, pero no, existen lenguajes de programación por debajo de éste. Cada vez que se le da una instrucción al procesador (como una MOV) como una secuencia de números binarios en lenguaje máquina, el microprocesador ejecuta un pequeño programa que lleva incorporado (cableado) escrito en lo que se llama *micro-código*. Para distinguir entre estos dos niveles llamaremos *macro-código* al código máquina, y micro-código al de nivel inferior siguiente. El micro-código es en realidad una especie de lenguaje máquina *para* la CPU *en* la CPU. El micro-código se guarda en una ROM *dentro* de la CPU. Un programa típico en micro-código que ejecutará una transferencia memoria-a-registro (instrucción MOV) comenzaría colocando la dirección de la fuente de información sobre el bus de direcciones; enviaría a continuación las señales de control necesarias para una lectura en memoria por el bus de control, recibiría sobre el bus de datos el dato leído y, finalmente, pondría el dato en el registro interno seleccionado. Tras este «programa», el procesador ejecutaría otro programa en micro-código para obtener la instrucción siguiente. Cada programa en micro-código requiere varios ciclos de reloj.

En algunos microprocesadores como el Motorola MC68000 hay incluso un nivel más bajo de programación; el *nano-código*. En este nivel, cada instrucción en micro-código se ejecuta a través de un pequeño programa en nano-código. De este modo se facilita cualquier modificación del conjunto de instrucciones de un microprocesador. Cada programa en micro-código consta de una serie de punteros a varios programas en nano-código dentro del procesador (véase la figura 2.27).

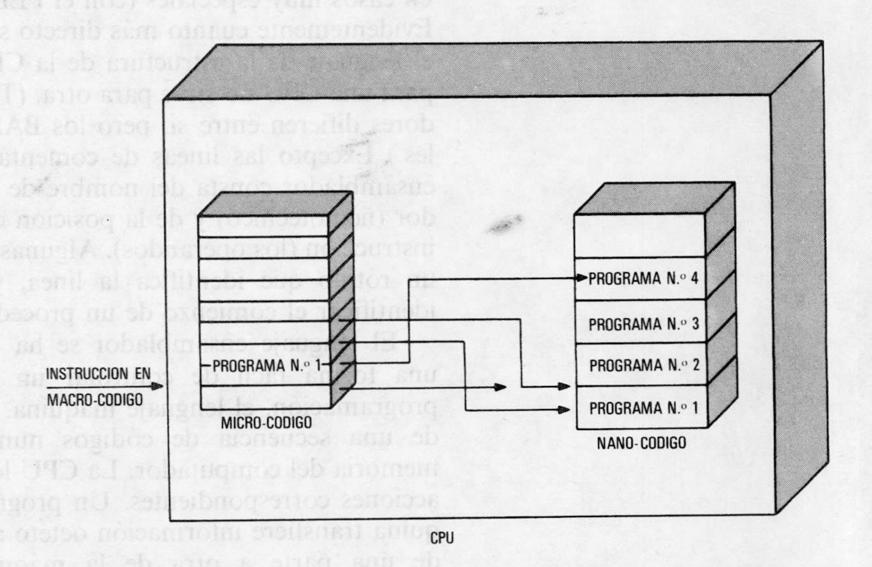


Figura 2.27: Micro-código y nano-código.

El micro-código (o el nano-código en el caso de Motorola) se compone normalmente de una serie de nano-códigos de instrucción. Cada nano-código de instrucción es un conjunto de bits con unos valores particulares; bastante más bits de los que se usan en el nivel superior inmediato, el macro-código. Desde un punto de vista muy general, cada bit del nano-código de instrucción corresponde a una señal de control de la CPU (véase la figura 2.28) que controla acciones como la carga de un registro desde el bus, operaciones aritméticas, carga de una dirección sobre el bus de direcciones, etc. Puesto que cada bit corresponde a una señal de control distinta, se pueden realizar simultáneamente varias de las acciones antes citadas. Las cosas no son tan simples en realidad. Ciertas señales de control se codifican juntas (se agrupan y tratan como un único número) para que ocupen un número menor de bits. En este caso, las señales de control que caen dentro del mismo grupo no pueden activarse simultáneamente ya que no se les puede seleccionar a la vez; pero dos señales que pertenezcan a grupos distintos sí pueden activarse al mismo tiempo.

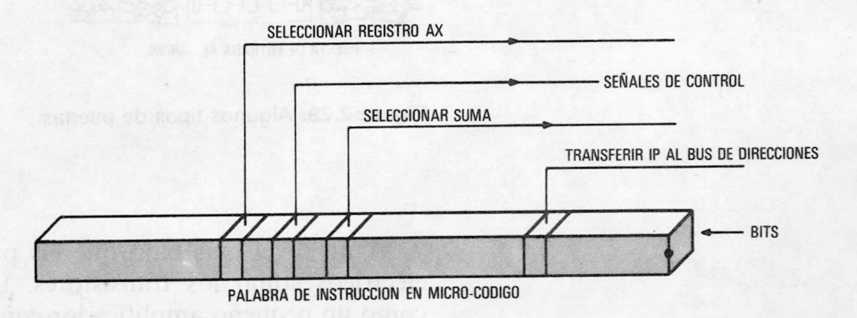


Figura 2.28: Micro-código y señales de control.

Dentro de la CPU, las señales de control van a parar a ciertas *puertas,* que son las unidades lógicas básicas de la computadora. Las puertas realizan todas las funciones lógicas básicas: AND, OR, XOR (OR exclusiva) y NOT. Toda puerta tiene una o más líneas de señal como entradas y una línea de salida (véase la figura 2.29). A su vez, circuitos formados por una docena o más de estas puertas realizan funciones lógicas algo más complejas; registros, circuitos de codificación y decodificación, funciones aritméticas y funciones lógicas. A estos circuitos llegan las señales de control que le dicen cuál de estas funciones debe realizar. Cada puerta puede tener tres posibles valores de salida: verdadero, falso y desconectada. El último estado recibe el nombre de «tercer estado», o «estado de alta impedancia», y permite que la puerta aparezca como si su salida no estuviera conectada al sistema. Cualquier señal que llegue a una puerta en tal estado queda completamente aislada de manera que no puede causar ningún conflicto con otras señales del sistema.

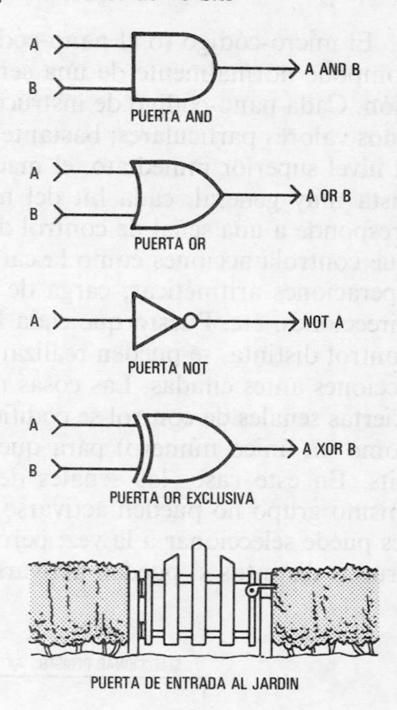


Figura 2.29: Algunos tipos de puertas.

A un nivel más bajo que las puertas están los componentes eléctricos como los transistores. Un transistor puede pensarse como un pequeño amplificador con varias señales de entrada que controlan la salida. Cada puerta contiene varios transistores. Por debajo de los transistores está ya la física de los semiconductores, con toda su familia de moléculas, átomos, electrones, quarks, etcétera, que todavía no hemos aprendido a programar.

DESARROLLO DE PROGRAMAS EN ENSAMBLADOR PARA PROCESADORES DE 16 BITS

Desarrollar programas en lenguaje ensamblador es algo más complicado que si se utiliza un intérprete BASIC, pero los resultados también son más interesantes. El proceso es casi idéntico al de desarrollar programas que utilicen compiladores de lenguajes como el FORTRAN, BASIC o Pascal. De hecho, el lenguaje ensamblador es algo íntimamente relacionado con los compiladores. La gran ventaja de los procesadores de 16 bits es que poseen un ensamblador con un nivel más alto que los antiguos ensambladores de las máquinas de 8 bits. La tendencia actual es desarrollar procesadores que se muevan a un nivel incluso más alto; por ejemplo, el iAPX 432 (32 bits), que tiene como lenguaje ensamblador un lenguaje de alto nivel, el Ada.

Cuando se quieren desarrollar programas en ensamblador, el primer paso consiste en escribir el llamado *código fuente*. El código fuente consta de una serie de nemotécnicos (nombres de instrucciones) y operandos (referencias a datos) seleccionados por

el programador para realizar ciertas acciones. El código fuente se escribe normalmente con la ayuda de un programa especial, el *editor de textos,* y se guarda, bien sea en memoria o sobre una cinta en un fichero que recibe el nombre de *fichero fuente.*

El paso siguiente consiste en traducir este código fuente al llamado *código objeto* que contiene ya los códigos numéricos reales para la máquina. Este proceso de traducción se realiza con la ayuda de un programa llamado *ensamblador* (no confundir con el lenguaje ensamblador).

Frecuentemente se necesita un paso más en el cual se combinan los varios módulos que forman el programa para formar el *fichero de carga.* Este proceso, el *encadenamiento* se realiza con la ayuda de un programa *editor de enlaces o de encadenamiento* (linker).

El último paso consiste en cargar el código máquina en memoria y ejecutarlo. Se realiza bien sea a través del sistema operativo, bien sea con la ayuda de un nuevo programa, el *cargador*.

Formato del código fuente

El código fuente puede aparecer en varios formatos dependiendo del ensamblador que se utilice. En general, los códigos fuente de los programas escritos en lenguaje ensamblador se organizan, cuando se imprimen o visualizan por pantalla, en cuatro columnas. La primera contiene los *rótulos* que definen posiciones de memoria (relativas) utilizadas por los saltos y llamadas a sub-rutinas, o simplemente como identificadores de posiciones de datos particulares. La segunda columna contiene los nemotécnicos (nombres de instrucciones), que reciben el nombre de *códigos de operación.* La tercera columna contiene los *operandos,* y la última columna se reserva para *comentarios.* Las columnas dividen a las líneas en zonas llamadas *campos.* Así, una línea típica consta de cuatro campos: el *campo de rótulos,* el *campo de código de operación,* el *campo de operandos* y el *campo de comentarios.* No todas las líneas tienen obligatoriamente la misma estructura. Veremos en concreto cómo ciertas líneas pueden estar formadas por un único campo, el campo de comentarios.

Cuando el programador en lenguaje ensamblador introduce en la máquina su código fuente, normalmente no tiene que preocuparse de que cada campo comience exactamente en la columna correcta. Tanto el ensamblador como los programas editores se encargan de ello, de manera que el programador sólo debe preocuparse de separar un campo de otro mediante un *delimitador* (como puede ser un blanco, o un tabulador). Además de las instrucciones que se traducirán al lenguaje máquina, el código fuente en lenguaje ensamblador contiene instrucciones especiales para el ensamblador. Dichas instrucciones reciben el nombre de *pseudo-instrucciones de ensamblador* o *pseudo-operaciones,* y se encargan de decirle al ensamblador la memoria que debe reservar para los datos, las posiciones en las que deben estar los tabuladores, etc.

Funcionamiento del ensamblador (en procesadores de 16 bits)

Veamos con más detalle qué hace un ensamblador. El proceso que realiza el ensamblador en un procesador de 16 bits es esencialmente el mismo que en el caso de procesadores de 8 bits, como mínimo desde el punto de vista del usuario. Por lo que respecta a la computadora, el proceso es incluso más sencillo debido a que, conceptos difíciles de implementar en los microprocesadores de 8 bits son relativamente fáciles para los procesadores de 16 bits. Realmente, lo que debería sorprendernos es que con un procesador de 8 bits se puedan hacer ciertas cosas como tener códigos objetos reubicables.

Normalmente, el ensamblador procesa dos veces el código fuente. En una primera pasada construye las tablas de referencia a direcciones (tablas de símbolos), y en la segunda pasada es cuando realmente traduce el código.

El código fuente contiene normalmente numerosas referencias entre partes del código. Para traducir correctamente el código, el ensamblador necesita construirse unas tablas que resuman dichas referencias. Estas tablas reciben el nombre de *tablas de símbolos,* porque, de hecho, guardan información de todos los símbolos del programa (identificadores) y de la posición de memoria en la que comienzan. Las referencias a posiciones de memoria que caen dentro del módulo que se está ensamblando reciben el nombre de *referencias locales,* en contraposición a las *referencias externas,* que involucran posiciones de memoria fuera del módulo en curso. Las referencias locales se resuelven durante el proceso de ensamblaje, pero las referencias externas necesitan un trato especial. Dentro del módulo en curso, puede haber rótulos, definidos de una forma especial, que pueden ser utilizados por otros módulos. Tales rótulos se declaran como *públicos* o *globales.* El ensamblador guarda estos rótulos en unas tablas específicas llamadas *tablas de símbolos globales,* e introduce en el código objeto ciertas instrucciones llamadas *instrucciones de encadenamiento,* que se encargan de pasar al editor de enlaces dichas tablas.

En la segunda pasada se escribe el código objeto en un *fichero objeto.* Al mismo tiempo que se carga el fichero objeto, se genera un *fichero listado* que contiene a la vez el código fuente y el correspondiente código máquina. El código fuente suele aparecer a la derecha, y el código máquina a la izquierda. Tras el listado de los dos códigos aparece la tabla de símbolos.

Las dos pasadas acarrean al ensamblador una cantidad de trabajo similar. Aunque la segunda fase parezca más sencilla, al no tener todas las instrucciones el mismo número de octetos en código máquina (y por tanto en memoria), el ensamblador debe examinar cuidadosamente cada instrucción antes de procesarla.

Tipos de código objeto

Hay dos tipos de código objeto. El primero de ellos es el *código absoluto.* El código absoluto es en el fondo un *código máquina,* directamente almacenable en memoria y preparado para ejecutarse. Dicho de otra manera, el código absoluto no necesita de ningún editor de enlaces. El código absoluto puede guardarse en varios formatos. El ensamblador ASM, que es parte del

sistema operativo CP/M, produce un formato particular del código absoluto que estudiaremos un poco más adelante. El segundo tipo de código objeto es el *código reubicable*. El código reubicable sí debe ser procesado por un editor de enlace antes de su ejecución. Es el tipo de código que produce el XM86, el ensamblador cruzado de Microsoft para el 8086. En el capítulo 7 veremos cómo funcionan el ASM y el XM86 sobre algunos programas-ejemplo escritos en lenguaje ensamblador. Utilizaremos el ASM para generar el programa «inicializador» de estos ejemplos, y el XM86 para ensamblar realmente el código del 8086.

Veamos ahora con un poco más de calma el tipo de código producido por el ASM. Está codificado en lo que se llama formato Intel HEX. Las principales características de este formato son que 1) utiliza el código ASCII para codificar su información, 2) incluye *direcciones de carga* de sus datos y 3) incluye octetos de verificación para la detección y corrección de errores. Los ficheros, visualizados sobre pantalla, o impresos sus listados, pueden entenderse fácilmente (por un usuario más o menos experto), al estar enteramente en código ASCII. El formato Intel HEX se puede utilizar para guardar cualquier tipo de datos que tenga que cargarse en memoria. Un fichero HEX se divide en registros. Cada uno de ellos corresponde a una línea del texto, y contiene un cierto número de octetos de datos y la dirección de carga de dichos datos. La dirección de carga indica la posición de memoria donde se guardará el primer octeto del registro; los demás se cargarán en posiciones consecutivas. Puesto que cada registro lleva una dirección de carga, un fichero HEX (que contiene varios registros) puede tranquilamente contener datos que se almacenen en posiciones muy dispares de memoria, aunque, eso sí, en posiciones perfectamente determinadas.

Es interesante conocer cómo guarda el fichero HEX toda esta información. Cada registro del fichero HEX comienza por dos puntos (:), a continuación lleva un número hexadecimal de dos dígitos (ASCII) que especifica el número de octetos de datos que contiene el registro; le sigue un nuevo número hexadecimal ahora de cuatro dígitos que da la dirección de carga; un campo de dos dígitos para el tipo de fichero que casi nunca se usa; los octetos de datos (cada uno codificado como un número hexadecimal de dos dígitos), y finalmente dos dígitos de control para la detección de errores. Es importante darse cuenta que, de nuevo, los octetos de datos del fichero HEX pueden cargarse en memoria directamente, sin ningún otro proceso intermedio salvo la conversión de hexadecimal a binario.

Además del formato Intel HEX existen otros formatos de almacenamiento de código máquina. En algunos, las direcciones de carga, los datos y los octetos de verificación se representan directamente en binario en vez de utilizar el código ASCII y el sistema de numeración hexadecimal.

El sistema operativo CP/M puede convertir los ficheros HEX a lenguaje máquina binario de dos formas distintas. La primera

utiliza una herramienta de depuración dinámica (DDT: Dynamic Debugging Tool) que pasa cualquier fichero HEX a binario y lo carga directamente en las posiciones de memoria designadas. La segunda pasa por un programa LOAD que, en vez de cargar los números binarios en memoria, genera un fichero con ellos. En el capítulo 7 veremos cómo utilizar el DDT para cargar y pasar control a nuestros programas-ejemplo.

Veamos ahora el segundo tipo de código objeto, el *código reubicable.* Es el tipo de código que producen los ensambladores cruzados de Microsoft, incluyendo el ensamblador cruzado del 8086 que utilizaremos en el capítulo 7. Los ficheros de este tipo de código deben combinarse vía el editor de enlaces con otros ficheros (que a su vez también están en el mismo tipo de código objeto reubicable) para formar el definitivo código máquina. En este caso es el editor de enlaces y no el ensamblador el que, después de juntar todas las piezas, determina la dirección de memoria en la que se cargará el código reubicable. Quizás sería más correcto llamar a este tipo de código «ubicable» mejor que «reubicable», ya que en realidad el código objeto no está «ubicado» inicialmente (a menos que el programador lo defina de un modo absoluto). El código reubicable contiene dos tipos de información: los octetos de código máquina y algunas instrucciones especiales para el editor de enlaces. En estos ficheros, el código máquina está incompleto debido a que las referencias de direcciones, o bien se han perdido, o bien están incompletas. Las instrucciones para el editor de enlaces son necesarias para que éste pueda completar dichas referencias de direcciones del código máquina. Estas instrucciones le informan de «detalles» tales como los *puntos de entrada, áreas comunes, símbolos externos* y tamaño del programa. Con estas informaciones, el editor de enlaces construye unas tablas con las que completa la información perdida.

Uso del editor de enlace

El disponer de un *editor de enlaces* (linker) además del ensamblador resulta muy interesante en aquellos casos en que el programa se ha escrito en distintos lenguajes de programación. Por ejemplo, un programa que se haya escrito básicamente en FORTRAN, pero que utilice rutinas escritas en lenguaje ensamblador. Lo mismo podría pasar con otros lenguajes de alto nivel como BASIC, Pascal y PL/1.

Cuando se utiliza esta técnica de mezclar más de un lenguaje de programación, el programa se divide en *módulos,* que realizan tareas específicas. El código fuente de cada módulo se escribe en un lenguaje particular y, posteriormente, se traducen todos ellos a un lenguaje común: el código objeto reubicable. Para los módulos escritos en lenguaje ensamblador, el traductor es un *ensamblador*. Para los módulos en FORTRAN u otros lenguajes de alto nivel, el traductor recibe el nombre de *compilador*. El resultado de la traducción es un conjunto de «trozos» de programa en código objeto. El editor de enlaces se encarga de «reunir» estos trozos para obtener el programa final. De nuevo, las posiciones reales d

memoria que ocupará el código máquina no se tienen en cuenta durante la escritura del código fuente, y ni tan siquiera se conocen en la traducción a código objeto. De hecho, algunos de los módulos se pueden obtener de unos ficheros especiales llamados librerías. Los módulos de una librería son programas de uso general en formato reubicable.

Evidentemente, dos módulos que estén simultáneamente en memoria no pueden solaparse a riesgo de que ocurran verdaderas catástrofes. Sin embargo, mientras se está ejecutando un programa, una misma área de memoria, a lo largo del tiempo, puede utilizarse para guardar datos de módulos distintos. Este concepto recibe el nombre de *solapamiento*. Se puede forzar el solapamiento especificando direcciones *absolutas* bien en el *código fuente,* bien durante el proceso de encadenamiento, ¡pero hay que vigilar cuidadosamente que no se solapen módulos! En caso de duda, la solución más recomendable es tomárselo con calma, trabajar en formato reubicable y dejar que el editor de enlaces se encargue de estas tareas.

Una de las razones principales por la cual el editor de enlaces es imprescindible en estos programas multi-módulo es que debe gestionar las referencias entre módulos distintos. Por ejemplo, varios módulos pueden compartir los mismos datos; o un cierto módulo puede ser llamado desde otros módulos distintos. Parte de las instrucciones del editor de enlaces le dicen *dónde* ocurren esas referencias en el código máquina «incompleto» y, si todo va bien, el resto de las instrucciones le dicen *cómo* completar la dirección para cada referencia. Puesto que el «dónde» está en un módulo, y el «cómo» suele venir determinado por otros módulos, un ensamblador que sólo es capaz de mirar un módulo a la vez es claramente insuficiente.

El resultado final producido por el editor de enlaces es el lenguaje máquina completo. El código máquina se guarda en un fichero llamado *fichero de carga*. Cada vez que se quiere ejecutar el programa, se pide al sistema operativo que cargue dicho fichero en memoria, y se comienza la ejecución. El editor de enlaces de Microsoft, el llamado L80, corriendo sobre CP/M guarda los ficheros de carga en binario, sin direcciones de carga. El formato es exactamente el mismo que el que produce el programa LOAD o la orden SAVE del CP/M.

Convenios sobre los nombres de los ficheros

En una aplicación particular pueden generarse varios ficheros, de modo que es importante tener alguna manera de distinguir estos ficheros para que tanto el usuario como el sistema operativo puedan identificar la aplicación particular y el tipo de código en el que están. Algunos sistemas operativos en disco, como CP/M de Digital Research, o RSTS y RSX de Digital Equipment permiten que el nombre del fichero conste de varias partes. Puede haber como máximo cuatro partes, pudiéndose omitir cualquiera de ellas. La primera parte es el número de cuenta, *imprescindible* en sistemas multi-usuario, en los cuales cada usuario tiene su o sus números de cuenta particulares. Se escribe entre paréntesis, por

ejemplo (178,34), para aislarlo del resto del nombre. La siguiente parte es el nombre del dispositivo del que procede. Suele venir a continuación del número de cuenta y antes del resto del nombre, y acabar en :. Así, KB: se refiere al teclado (KeyBoard); DKO: a la unidad de discos 0 del PDP-11, y A: a la unidad de discos A en CP/M. La tercera parte es la parte principal del nombre del fichero. Puede estar formado por cualquier combinación de letras y números en un número limitado (8 para el CP/M). En algunos sistemas operativos, el nombre no puede comenzar por un número, y en otros pueden incluso utilizarse los símbolos especiales. La parte principal del nombre sirve para identificar el fichero con independencia de su forma y formato. (Un poco más adelante se sigue hablando de este tema). La última parte del nombre es la extensión. Las extensiones de fichero se reconocen y generan mediante ciertos programas del sistema operativo, incluidos compiladores, ensambladores y editores de enlace. Normalmente indica el tipo o formato del fichero, y viene separado del resto del nombre por un punto.

Para concretar ideas, veamos algún ejemplo:

B:SORT.FOR podría ser el nombre de un fichero en CP/M, localizable en la unidad de discos flexibles B. La extensión del fichero, FOR, indica que su código fuente está en FORTRAN. La parte principal del nombre, SORT, identifica el nombre del programa FORTRAN. La parte principal del nombre suele aportar alguna pista al proceso que realiza el programa.

A:L88.ASM podría ser un código fuente ensamblador de un módulo de un programa llamado L88, almacenado en la unidad de discos A del sistema operativo CP/M. A menos que se indique lo contrario, el sistema operativo CP/M supone por omisión la unidad de discos A. De igual manera podríamos haber escrito simplemente L88.ASM como nombre del fichero. L88 es el nombre de uno de los ficheros (el programa inicializador) que veremos en el capítulo 7. La extensión ASM se utiliza en ficheros en código fuente del ensamblador ASM del sistema.

L88.HEX es el nombre de un fichero objeto (en la unidad A por omisión) producido al ensamblar el fichero L88.ASM anterior. La extensión del fichero HEX la incluye automáticamente el ensamblador ASM en el nombre de todos los ficheros en código objeto (absoluto) que produce. El ensamblador ASM genera también otro fichero, el L88.PRN, que recibe el nombre de fichero listado. Este fichero contiene el código máquina y el código fuente del programa simultáneamente, para facilitar su comparación.

L88.COM es el nombre del fichero generado por la orden LOAD actuando sobre el fichero L88.HEX anterior. Cuando el programa LOAD (del CP/M) transforma el fichero HEX a formato binario, se genera un fichero con el mismo nombre principal pero con la extensión COM. Cualquier fichero con esta extensión puede utilizarse exactamente como si fuese una orden. En nuestro caso, dispondremos de una nueva orden L88 que nos será muy útil en la inicialización del código 8088.

Ensambladores y compiladores Microsoft

Microsoft ofrece un paquete FORTRAN y BASIC preparado para funcionar sobre CP/M. Incluye un ensamblador y un editor de enlaces. Bajo estos sistemas, la extensión FOR indica ficheros en código fuente FORTRAN; BAS indica ficheros código fuente BASIC; MAC designa los ficheros código fuente en lenguaje ensamblador del ensamblador MACRO8080; REL designa los ficheros objetos MACRO reubicables producidos por estos compiladores y el ensamblador, y COM designa los ficheros binarios absolutos generados por el editor de enlaces.

REL quiere significar código objeto REubicabLe, el segundo tipo de código objeto. Los ficheros COM son compatibles con los ficheros COM generados por el programa LOAD.

Microsoft tiene también un ensamblador para el 8086/8088 llamado XMACRO-86. El XMACRO-86 funciona sobre el sistema operativo CP/M-80 utilizado por las CPU 8080, 8085 y Z80. Como este ensamblador produce códigos para CPU diferentes a sobre la que está actuando, se le da el nombre de *ensamblador cruzado*. El ensamblador cruzado de Microsoft acepta ficheros fuente con la extensión MAC (creados mediante editor), y produce ficheros objeto con la extensión REL. Estos ficheros REL pueden ser procesados por el mismo editor de enlaces (L80) trabajando con el ensamblador MACRO8080. Como se puede observar, el editor de enlaces no necesita saber nada sobre el lenguaje máquina excepto la información sobre el direccionamiento tal como viene especificado por las instrucciones contenidas en el fichero REL. Esto es, el resto del código máquina es un dato más para el editor de enlaces. En los ejemplos del capítulo 7 utilizaremos este ensamblador cruzado y el editor de enlaces. Microsoft tiene preparada una nueva versión del ensamblador para el 8086/8088 que trabajará sobre el CP/M-86. De esta manera se espera que trabaje incluso mejor que si todo el proceso completo se hiciese sobre un solo procesador.

Programa-ejemplo en lenguaje ensamblador de 16 bits

Veamos un pequeño programa. Presentaremos primero la versión BASIC, para después traducirlo a lenguaje ensamblador de nivel más bajo, y ver cómo se puede modificar para mejorar su eficiencia, uno de los aspectos más interesantes de la programación de los microprocesadores de 16 bits.

Comenzaremos con el programa BASIC. Se trata de escribir un programa que genere e imprima la sucesión de Fibonacci:

```
1, 1, 2, 3, 5, 8, 13 ...
```

Cada número de la sucesión es la suma de los dos anteriores. La sucesión de Fibonacci tiene muchas aplicaciones en las artes y en las ciencias, en temas tan dispares como las escalas musicales, el crecimiento de hojas en los árboles o ¡la reproducción de conejos!

Veamos primero el código fuente BASIC:

```
100 REM - FIBONACCI
110 REM
120 X% = 0
130 Y% = 1
140 PRINT Y%;
150 Z% = X% + Y%
160 PRINT Z%;
170 X% = Y%
180 Y% = Z%
190 GOTO 150
```

Figura 2.30: Programa BASIC de la sucesión de Fibonacci.

Casi todos los compiladores poseen una opción que permite traducir el código fuente de un programa escrito en un lenguaje de alto nivel a lenguaje ensamblador. Exceptuando cómo se gestiona la salida, y algunos procedimientos especiales de tratamiento de errores, la traducción del código fuente en el 8086/8088 a lenguaje ensamblador es típica:

```
; Sucesión de Fibonacci
;
X        DW      0        ; Almacenamiento para X
Y        DW      0        ; Almacenamiento para Y
Z        DW      0        ; Almacenamiento para Z
;
L00100:  ; Comienzo del programa
L00110:
L00120:  MOV     X,0      ; Poner X = 0
L00130:  MOV     Y,1      ; Poner Y = 1
L00140:  MOV     AX,Y     ; Obtener Y
         CALL    OUTPUT   ; Imprimirlo
;
L00150:  MOV     AX,X     ; Obtener X
         ADD     AX,Y     ; Sumarle Y
         MOV     Z,AX     ; Guardar el resultado
L00160:  MOV     AX,Z     ; Obtenerlo
         CALL    OUTPUT   ; Imprimirlo
L00170:  MOV     AX,Y     ; Cargar Y en X
         MOV     X,AX
L00180:  MOV     AX,Z     ; Cargar Z en Y
         MOV     Y,AX
L00190:  JMP     L00150   ; Término siguiente
```

Figura 2.31: Traducción del programa BASIC a lenguaje ensamblador.

La impresión de resultados se lleva a cabo en una rutina OUTPUT, disponible en algún lugar de memoria. Las instrucciones MOV, ADD, CALL y JMP del 8086/8088 utilizadas en el ejemplo se estudiarán en detalle en el capítulo siguiente. Cada línea de instrucción genera varios octetos de código máquina.

La instrucción DW que aparece en las tres primeras líneas no es una instrucción de la CPU sino una instrucción para el ensamblador (pseudo-operación) que le indica que debe reservar una celda de memoria (en este caso una palabra=2 octetos) para la variable indicada.

Aquí se reserva una palabra para cada variable X, Y y Z. Notemos que hay rótulos un tanto extraños. Corresponden, con

un formato más cercano al código máquina, a los números de sentencia del programa BASIC. Por ejemplo, L00150 corresponde a la línea número 150 del programa BASIC.

El programa sigue el mismo *algoritmo* (método de cálculo) que el programa BASIC, moviendo palabras entre las posiciones de memoria X, Y, Z, y el registro AX de la CPU.

Este sistema de programar en lenguaje ensamblador a base de programar en un lenguaje de alto nivel y luego traducirlo, no es en absoluto el más recomendable. El programa resultante es bastante ineficaz desde el momento que el compilador realiza fundamentalmente una traducción línea-a-línea, en vez de global. Utilizando otros registros de la CPU se puede escribir una versión mejor. Utilizaremos el registro BX para la variable X, el registro CX para la Y, y el AX para la Z, en vez de poner las variables en RAM. El programa quedaría así:

```
; Sucesión de Fibonacci
;
ENTER: MOV    BX,0      ; Poner X = 0
       MOV    CX,1      ; Poner Y = 1
       MOV    AX,CX     ; Obtener Y
       CALL   OUTPUT    ; Imprimirlo
;
LOOP:  MOV    AX,BX     ; Obtener X
       ADD    AX,CX     ; Z = X + Y
       CALL   OUTPUT    ; Imprimirlo
       MOV    BX,CX     ; Cargar Y en X
       MOV    CX,AX     ; Cargar Z en Y
       JMP    LOOP      ; Término siguiente
```

Figura 2.32: Programa en lenguaje ensamblador optimizado.

Aprovechando algunas instrucciones particulares del 8086/8088 se puede conseguir un programa todavía mejor. La instrucción XCHG reduce el número de pasos del programa, haciéndolo más rápido, y disminuyendo la cantidad de memoria utilizada.

```
; Sucesión de Fibonacci
;
ENTER: MOV    BX,0      ; Poner X = 0
       MOV    AX,1      ; Poner Y = 1
       CALL   OUTPUT    ; Imprimirlo
;
LOOP:  XCHG   AX,BX     ; Intercambiar X e Y
       ADD    AX,BX     ; X = X + Y
       CALL   OUTPUT    ; Imprimirlo
       JMP    LOOP      ; Término siguiente
```

Figura 2.33: Programa en lenguaje ensamblador optimizado.

CONCLUSIONES

En este capítulo hemos introducido algunos conceptos básicos sobre los microprocesadores de 16 bits que nos permitirán comenzar nuestro estudio de los procesadores Intel 8086, 8088, 8089 y 8087. Hemos dado una introducción somera a temas como la organización de una microcomputadora moderna, y cómo los chips simplifican su diseño; la segmentación, el multi-proceso, el lenguaje máquina, el micro-código e incluso el nano-código. Esto nos permitirá comprender mejor todas aquellas características de los nuevos microprocesadores de 16 bits que no poseían los de 8.

Hemos visto y aplicado el principio «filosófico» de la ortogonalidad a diversos temas como el bus S-100, el almacenamiento de datos, y en el contexto del multi-proceso.

En los capítulos inmediatos presentaremos un estudio más detallado de cada uno de los chips. Comenzaremos en el capítulo 3 con las dos unidades centrales de proceso de propósito general de la familia Intel iAPX-86. Veremos su relación con sus predecesores de 8 bits: el 8008, el 8080 y 8085, y pasaremos revista a sus principales características. En los capítulos 4 y 5 cubriremos dos procesadores de propósito especial: el Procesador de Datos Numérico 8087 y el Procesador de E/S 8089. Veremos cómo ambos procesadores pueden trabajar en conjunción con cualquiera de los dos procesadores Intel de propósito general, y los notables resultados que producen. En el capítulo 6 presentaremos algunos de los chips controladores de Intel y veremos cómo se pueden utilizar para distribuir tareas y aliviar a los procesadores periféricos y de propósito especial. En el capítulo 7 daremos un repaso al estado actual del tema, especialmente interesante por cuanto las grandes firmas en el mundo de las computadoras están entrando en el mercado de las computadoras personales ofreciendo máquinas que utilizan el 8086/8088.

Capítulo 3

Los procesadores de propósito general 8086 y 8088

El 8086 y el 8088 son microprocesadores de 16 bits de propósito general de Intel. Al igual que otros muchos fabricantes/diseñadores de chips microprocesadores de esta nueva generación, Intel ofrece varias versiones de sus productos. El 8086 es un microprocesador de 16 bits, tanto en lo que se refiere a su estructura como a sus conexiones externas, mientras que el 8088 es un procesador de 8 bits que ¡internamente es idéntico al procesador de 16 bits 8086! En este capítulo comentaremos las características más importantes de estos dos procesadores y cómo el 8088 de bus «restringido» se acopla en el esquema general del resto de microprocesadores de Intel.

Aunque Intel fue la primera en diseñar y fabricar los chips 8086 y 8088, hay otros fabricantes que producen su propia versión idéntica de estos chips. Estos fabricantes reciben el nombre de «segundas fuentes». Actualmente, Intel está alentando a estos fabricantes en el desarrollo de estas versiones idénticas. La razón es que algunos consumidores de chips (como el Gobierno Federal Americano), no diseñan ni construyen nada que use un chip del que sólo exista un proveedor. El 8086 y 8088 están siendo fabricados por gran número de empresas (la mayoría japonesas), incluyendo a Fujitsu. Intel, de hecho, garantiza a Fujitsu una licencia a nivel mundial para fabricar y vender los chips 8086, 8088 y 8089. La licencia no es exclusiva y no implica ningún intercambio de dinero entre las dos compañías. El resto de fabricantes de estos chips (incluyendo compañías como Siemens, AMD y NEC) operan sin ninguna licencia especial, pero con el beneplácito de Intel.

El 8086 y el 8088 son prácticamente idénticos excepto por el tamaño de su bus de datos externo. Intel trata esta igualdad interna y desigual de la externa, dividiendo cada procesador 8086 y 8088 en dos sub-procesadores. O sea, cada uno consta de una unidad de ejecución (EU: Execution Unit) y una unidad interfaz de bus (BIU: Bus Interface Unit). La unidad de ejecución está encargada de realizar todas las operaciones mientras que la unidad de interfaz de bus está encargada de acceder a datos e instrucciones del mundo exterior. Las unidades de ejecución son idénticas en ambos procesadores, pero las unidades de interfaz de bus son diferentes en varias cuestiones que trataremos más adelante. Esta aproximación es un ejemplo claro de diseño modular. Esto es, el todo (el procesador) se divide en partes (los

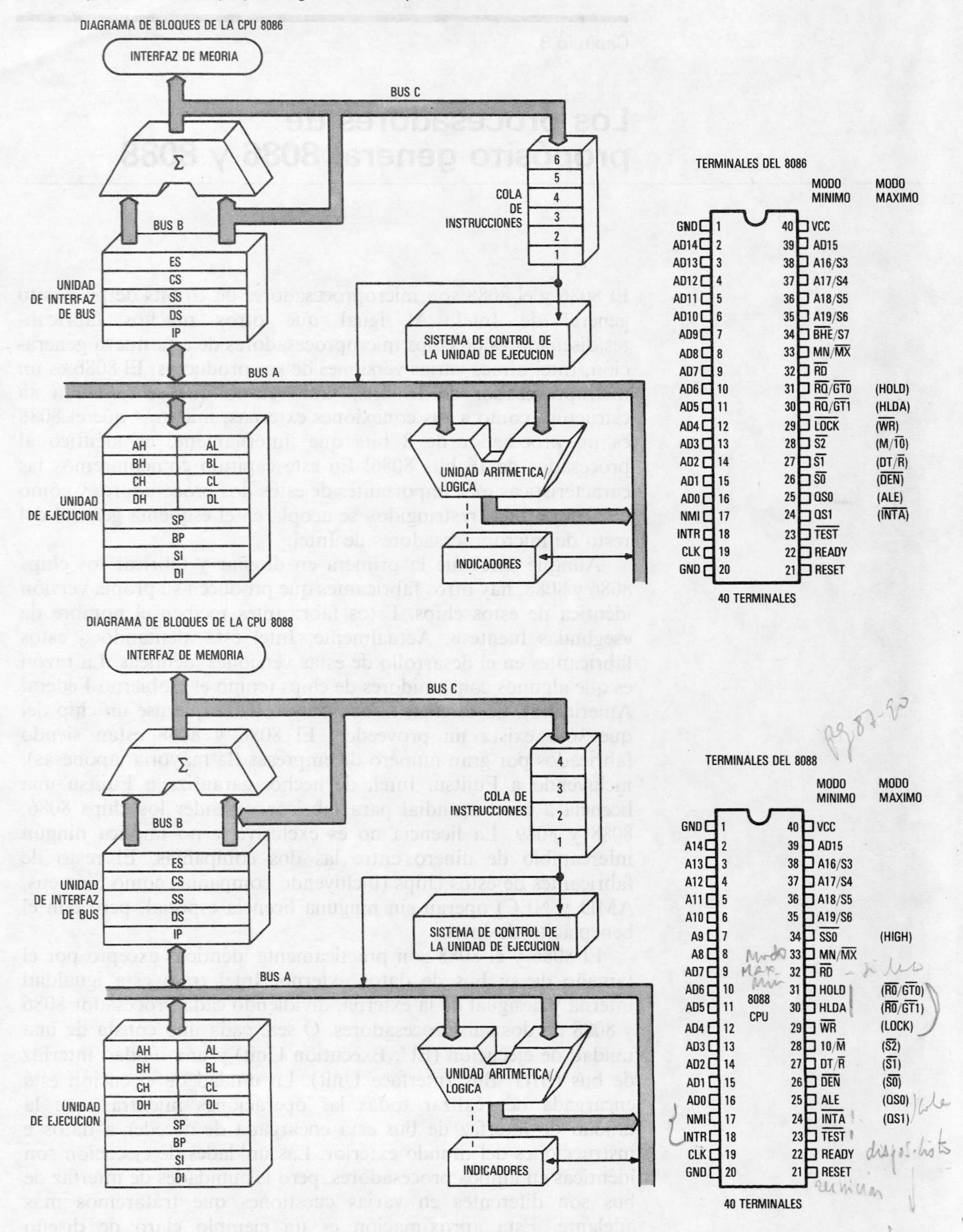


Figura 3.1: Los procesadores de propósito general 8086 y 8088.

dos subprocesadores), y cada parte o módulo forma una unidad de trabajo encargada de ciertas subtareas. Esto es un principio fundamental en la teoría moderna de diseño del hardware y software, así como también lo es en otros temas que no tienen nada que ver con computadoras. En este caso, la ventaja de esta aproximación modular es el ahorro de esfuerzo necesario para producir el chip 8088. Sólo una mitad del 8086 (el BIU) debe rediseñarse para producir el 8088.

¿POR QUE EL 8088?

Como hemos comentado hasta ahora, el 8088 tiene un bus de datos externo reducido (8 bits). La razón para crear el 8088 con este bus de datos reducido es prever la continuidad entre el 8086 y los antiguos procesadores de 8 bits de Intel, el 8080 y el 8085. Esta continuidad es especialmente importante para aquellos que han desarrollado investigaciones en estos tipos de productos. Teniendo el mismo tamaño de buses (así como requerimientos similares de control y tiempo), el 8088, que es internamente un procesador de 16 bits, puede reemplazar a uno de esos primeros procesadores de 8 bits en un sistema ya existente. Esto mejorará el rendimiento del sistema a un coste muy bajo. Por ejemplo, hay varias placas de CPU 8088 disponibles en estos momentos para el bus S-100. Si ya se tiene un sistema con un bus S-100, el coste de pasarse a uno de 16 bits (internamente) se limita justamente a una nueva placa de CPU (de 300 a 500 $) y un nuevo software que, incidentalmente, está llegando a ser más caro que el hardware.

Otra ventaja del 8088 es que los nuevos sistemas, pequeños y baratos aunque muy potentes, pueden diseñarse basándose en este chip. La buena razón *precio/rendimiento* de los sistemas basados en el 8088, hará que sea éste el elegido por todos aquellos que deseen diseñar computadoras fáciles de vender.

El primer microprocesador de 8 bits barato fue el 8080, que llegó a ser uno de los favoritos de los diseñadores, por encima de otros como el 6502 y 6800 que salieron al mercado casi al mismo tiempo y eran incluso más baratos. Se produjo una gran cantidad de hardware y software para sistemas basados en el 8080. Sin embargo, el 8080 tenía ciertas desventajas, como el requerir tres tensiones de alimentación y dos señales de reloj diferentes. Más tarde se diseñó el 8085 para soslayar estos problemas. El 8085 requería únicamente una tensión de alimentación y producía sus propias señales de reloj (hay terminales que permiten adaptar el cristal oscilador). Además, tenía una velocidad más alta que el 8080. Debido a estas mejoras, el 8085 no es directamente compatible, desde el punto de vista *hardware,* con el 8080, pero sí es, en un 99 por 100 de los casos, compatible desde el punto de vista software[1].

[1] La diferencia radica en dos instrucciones, la RIM (Real Interrupt Mask, Leer máscara de interrupción) y la SIM (Set Interrupt Mask, Activar máscara de interrupción), incluidas en el juego de instrucciones del 8085, pero no en el 8080. Además de sus funciones de lectura y activación de las máscaras de interrupción, estas instrucciones también pueden usarse para recibir y enviar datos bit a bit, en serie, por una línea especial de datos del 8085.

SIMILITUDES DEL 8088 Y DEL 8085

El 8088 tiene muchas señales en común con el 8085, particularmente las asociadas con la forma en que los datos y las direcciones están multiplexadas, aunque el 8088 no produce sus propias señales de reloj tal como lo hace el 8085. El 8088 y el 8085 siguen el mismo esquema de compartir los terminales correspondientes a los 8 bits más bajos del bus de direcciones con los 8 bits del bus de datos, de manera que bastan 16 terminales para direcciones y datos, en vez de los 24 previsibles.

El 8085 y el 8088 pueden, de hecho, dirigir directamente los mismos chips controladores de periféricos. Las investigaciones hardware para sistemas basados en el 8080 o el 8085 son, en su mayoría, aplicables a sistemas basados en el 8088.

Por supuesto que habrá problemas con la velocidad del sistema antiguo, pero uno siempre puede bajar la velocidad del 8088 instalándole el cristal oscilador apropiado.

Dado que el juego de instrucciones del 8086 y 8088 tiene una capacidad mayor que el del 8080 y 8085, la compatibilidad del software está lejos de ser directa. Los registros del 8080/8085 son un subconjunto de los registros del 8086/8088 y la mayoría de instrucciones del 8080/8085 pueden ser traducidas directamente a instrucciones del 8086/8088. Esto sin embargo, es mejor hacerlo

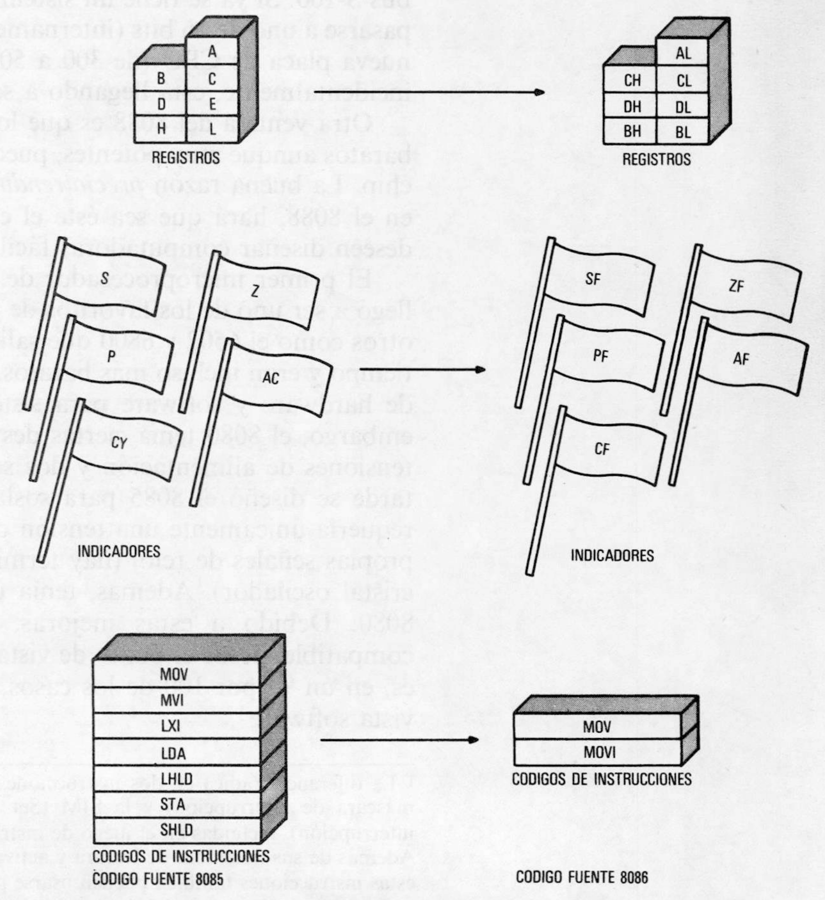


Figura 3.2: Traducción del código fuente del 8085 al 8086.

en lenguaje fuente que en lenguaje máquina, ya que éste es enteramente diferente de uno a otro. Intel ofrece un programa traductor de fuente 8080/8085 a fuente 8086/8088. Este programa debe realizar la traducción en varios pasos (véase la figura 3.2). Entre ellos están: 1) traducir el nombre de los registros: Ejemplo, los registros de 8 bits A, B, C, D, E, H y L pasan a ser AL, CH, CL, DH, DL, BH y BL, respectivamente; 2) traducir los indicadores: por ejemplo, S, Z, AC, P y CY pasan a ser SF, ZF, AF, PF y CF, respectivamente; 3) traducir las instrucciones: por ejemplo, MOV, MUI, LXI, LDA y STA pasan a ser, todas ellas, variaciones de las instrucciones MOV; 4) traducir los operandos: por ejemplo, un operando de 16 bits de la instrucción LXI del 8080/8085 pasa a ser una constante; un operando de 8 bits de la instrucción LDA, o uno de 16 bits de la instrucción LHLD del 8080/8085 pasan a ser una variable, y 5) traducir las pseudo-instrucciones del ensamblador. La entrada al programa traductor de Intel debe ser un código fuente compatible con el ensamblador Intel del 8085; la salida que dará el programa será un código fuente compatible con el ensamblador Intel del 8086. Normalmente, los códigos fuentes del 8085 y 8086 son compatibles con el ensamblador Intel. Sin embargo, existen fuentes que no lo son, como el ensamblador de Microsoft para el 8086, que usa nombres diferentes en algunas instrucciones y la gestión de los tipos de datos es diferente a la de Intel. Afortunadamente, Microsoft está creando un programa traductor para estas fuentes que pronto estará disponible.

PRINCIPALES CARACTERISTICAS DE LA CPU DEL 8086/8088: VISION GENERAL

En esta sección veremos las principales características de los chips de CPU 8086 y 8088. En las siguientes secciones nos extenderemos en los tópicos que veremos de forma general en esta sección, como las formas de direccionamiento, el juego de registros y el juego de instrucciones.

Capacidad de direccionamiento

Tanto el 8086 como el 8088 tienen un bus de direccionamiento de 20 bits de amplitud, lo que les provee la capacidad de direccionar un megaocteto de memoria. Para aquellos que les gusten las matemáticas esto es porque:

$2^{20} = 1.048.576 = 1$ megaocteto

Sin embargo, el registro de direccionamiento del 8086/8088, tiene únicamente una amplitud de 16 bits. Esto es equivalente a 64 K octetos. Este procesador usa un método llamado *segmentación* para permitir el direccionamiento a todo el megaocteto de memoria.

Más tarde, en este mismo capítulo, veremos cómo el 8086/8088 realiza la segmentación. Zilog tiene dos versiones del Z8000, una con segmentación y otra sin ella. La versión sin segmentación tiene únicamente 16 bits para direccionar, mientras que la versión con segmentación permite acceder a memoria con 24 bits de direccionamiento (16 megaoctetos), pero requiere el uso

de un chip especial llamado Unidad de Gestión de Memoria (MMU: Memory Management Unit). El Motorola MC68000 tiene 32 bits con sus registros de direccionamiento, pero sólo envía 24 bits al «mundo exterior». Obviamente, está diseñado para futuras expansiones, y al igual que el Z8000, el 68000 está diseñado para trabajar con una unidad de gestión de memoria. El 8086/8088 no necesita de ningún chip adicional para realizar la segmentación, y las versiones futuras (el iAPX 286) se están diseñando con un sofisticado esquema de gestión de memoria incorporado en conjunción con un direccionamiento de 24 bits.

El 8086/8088 tiene otra memoria separada, llamada espacio de E/S (véase la figura 3.3), que puede considerarse como una memoria extra para direccionar en la cual se encuentran los aparatos de E/S (cableadas las direcciones). En la Motorola MC68000 no existe espacio de E/S; todos los aparatos aparecen en la CPU localizados en la memoria. En el 8086/8088 (y en el Z8000), los aparatos pueden conectarse a la memoria principal o al espacio de E/S. Asimismo, la memoria puede conectarse a la memoria principal o al espacio de E/S. El espacio de E/S del 8086/8088 usa un direccionamiento de 16 bits (permite direccionar 64K octetos). Los anteriores procesadores de 8 bits de Intel tenían únicamente un direccionamiento de 8 bits para el espacio de E/S, lo que permitía direccionar 256 octetos de dicho espacio. Esto es, realmente, una cantidad muy pequeña. El direccionamiento actual de 16 bits, posibilita a Intel el poner muchas cosas en el espacio de E/S, incluyendo entre ellas, la memoria asociada para el IOP-8089, así como el controlador de aparatos de E/S. Se puede pensar que al poner toda la E/S en el espacio de E/S (incluyendo el IOP-8089 y todos los controladores de aparatos que se comunican con él) se elimina gran cantidad del tráfico entre la CPU y la memoria; pero no es cierto, ya que tanto la memoria principal como el espacio de E/S usan el mismo juego

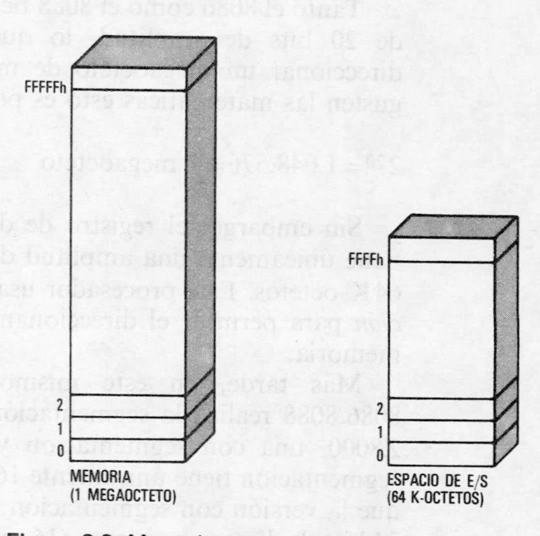


Figura 3.3: Memoria y espacio E/S.

de direcciones y líneas de datos. Sólo el uso de señales de control separa la memoria y el espacio de E/S. A pesar de ello es posible emplazar los controladores en buses separados, los cuales se conectarían al bus del procesador principal, con lo cual se reduciría el tráfico. Esto requeriría usar un chip de interfaz de bus como el que se explica en el capítulo 6.

Juego de registros

El juego de registros del 8086 y del 8088 son exactamente iguales, con 14 registros internos de 16 bits. Cada registro tiene su propia personalidad, aunque varios comparten tareas comunes. El juego de registros del 8086/8088 es una ampliación del juego de registros del 8080, que a su vez es una ampliación del juego de registros del 8008. En la figura 3.4 se puede comparar el juego de registros del 8008, 8080 y 8086.

Modalidades de direccionamiento

El 8086/8088 tiene 25 *modalidades de direccionamiento* diferentes; una modalidad de direccionamiento es un conjunto de reglas que especifican la localización (posición) de un dato usado

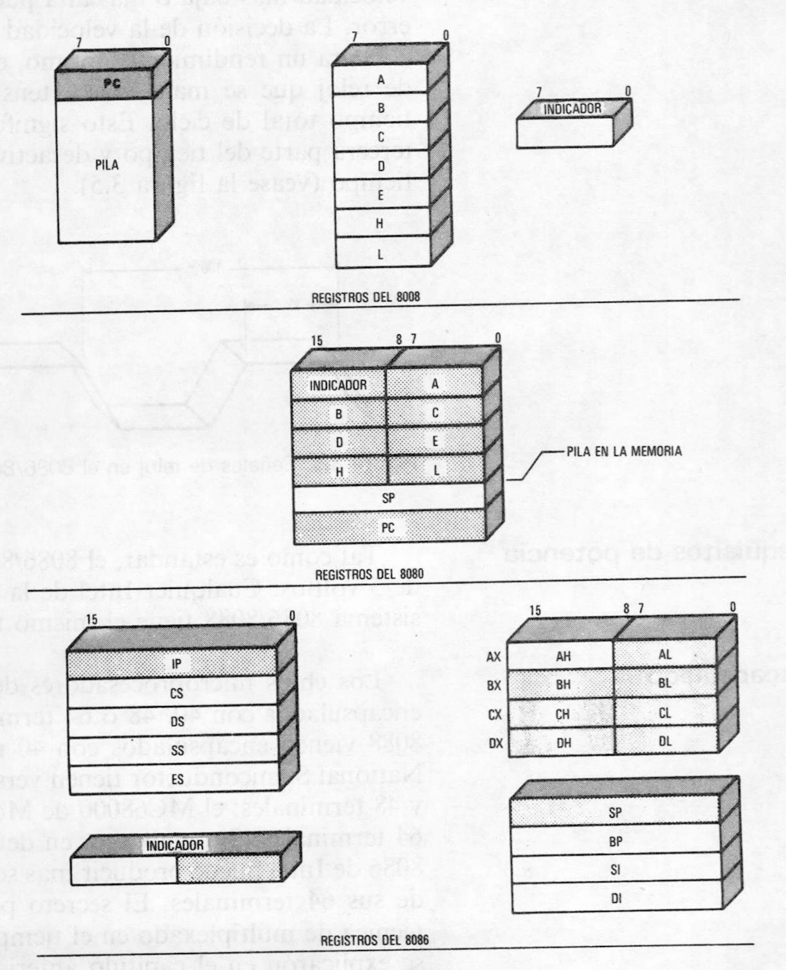


Figura 3.4: Comparación del juego de registros del 8008, 8080 y 8086.

durante la ejecución de una instrucción. En la modalidad más sencilla, un dato se localiza en un registro determinado; en la modalidad más compleja, se suma el contenido de dos registros en una cantidad de 8 ó 16 bits, que se encuentra en el programa. El resultado de la suma nos indica la dirección del dato. Desarrollaremos las modalidades de direccionamiento, más adelante en esta misma sección.

Señales de reloj

Al igual que los más recientes microprocesadores, el 8086/8088 requiere una única señal de reloj. Al contrario que el 8085, estos procesadores no generan su propia señal de reloj, dependiendo del generador de reloj 8284, que usa un cristal oscilador para determinar la frecuencia de señal. Intercambiando este cristal, se puede seleccionar diferentes velocidades de operación. Intel tiene una versión de 5 MHz y otra de 8 MHz para el 8086 y el 8088. Estas versiones representan las velocidades más altas, recomendadas para estos chips. No se recomienda bajar de 2 MHz para ninguno de los dos. Dependiendo de su chip particular, una velocidad más baja o más alta puede ser correcta o puede causar error. La decisión de la velocidad a adoptar sólo puede ser suya.

Para un rendimiento óptimo, el 8086/8088 requiere una señal de reloj que se mantenga a tensión alta una tercera parte del tiempo total de ciclo. Esto signifca que el reloj está activo una tercera parte del tiempo y desactivado las dos terceras partes del tiempo (véase la figura 3.5).

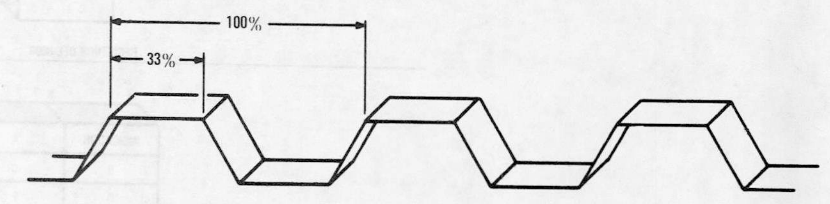


Figura 3.5: Señales de reloj en el 8086/8088.

Requisitos de potencia

Tal como es estándar, el 8086/8088 requiere una alimentación de 5 voltios. Cualquier Intel de la serie 8000 relacionado con un sistema 8086/8088 tiene el mismo tipo de requerimiento.

Encapsulado

Los chips microprocesadores de 16 bits generalmente vienen encapsulados con 40, 48 ó 64 terminales. Tanto el 8086 como el 8088 vienen encapsulados con 40 terminales. Tanto Zilog como National Semiconductor tienen versiones de sus productos con 40 y 48 terminales; el MC68000 de Motorola viene encapsulado con 64 terminales. Sin embargo, en determinadas configuraciones, el 8086 de Intel puede producir más señales que el MC68000 a pesar de sus 64 terminales. El secreto para poderlo lograr está en la técnica de multiplexado en el tiempo y la de codificación; ambas se explicaron en el capítulo anterior. Recordemos los conceptos básicos.

Multiplexado en el tiempo significa usar el mismo conjunto de líneas, pero en períodos de tiempo distintos, para enviar conjuntos de señales diferentes. *Codificación* significa convertir un conjunto de estados posibles en números, y enviar estos números por unas pocas líneas en vez de usar una línea para cada estado diferente. (Por ejemplo, ocho estados pueden ser codificados con números binarios del 0 al 7, los cuales pueden enviarse con tres únicas líneas.)

Multiproceso y procesamiento en paralelo

El 8086/8088 tiene unas características especiales, que permiten coordinar sus actividades con otros procesadores en contextos de *multiproceso* y de *procesamiento en paralelo*. El procesamiento (tratamiento) en paralelo es un sistema en el cual dos o más procesadores trabajan en tándem en la misma porción de un programa. Multiproceso es un sistema en el cual dos o más procesadores trabajan en diferentes programas, pero comparten la misma memoria. El NDP 8087 utiliza el procesamiento en paralelo y el IOP 8089 utiliza el multiproceso. Desarrollaremos estos conceptos al detallar el Procesador de Datos Numéricos 8087 y el Procesador de E/S en los dos próximos capítulos.

Juego de instrucciones

El juego de instrucciones del 8086/8088 de Intel es comparable al de los otros líderes del mercado: el MC68000 de Motorola y el Z8000 de Zilog. Los tres fabricantes han adoptado un amplio juego de instrucciones haciendo que sus máquinas de 16 bits sean mucho más potentes que sus equivalentes de 8 bits. Han añadido nuevas operaciones como la multiplicación y la división, y han incrementado la eficacia de las antiguas operaciones como la de saltar a una dirección calculada en el programa. Como veremos con más detalle en la sección dedicada al juego de instrucciones, es importante realzar que los procesadores de Zilog y Motorola son, realmente, en muchos aspectos, máquinas de 32 bits encapsuladas en chips de 16 bits. El 8086 de 16 bits y el 8088 de 8 bits pueden compararse de forma favorable, tanto en velocidad como en flexibilidad, con el Motorola o Zilog, en cuanto a las operaciones estándar de 8 y 16 bits. Sin embargo, por motivos como ser más lento en operaciones de multiplicar y dividir y la laguna de las operaciones directas con bits, el 8086, generalmente, se encuentra detrás de los otros dos en pruebas de velocidad. De todas formas, esto ocurre únicamente cuando en las pruebas no se incluyen los procesadores periféricos, el Procesador de E/S 8089 y el Procesador de Datos Numéricos 8087. Con estos periféricos el Intel 8086 forma una combinación mucho más potente que cualquiera de los otros dos, trabajando solos. Además, el 186, siguiente versión de 8086, está diseñado para aumentar las velocidades de las instrucciones que ahora bajan la media del 8086.

ARQUITECTURA TUBULAR (PIPELINE)

Dado que el bus de datos del 8086 es el doble de ancho que el del 8088, se podría esperar que el 8086 fuera el doble de rápido que el 8088. Esto no es verdad, por varias razones. Una razón es

que muchas aplicaciones necesitan transferir datos en 8 bits. Otra es que el procesador tiene que hacer bastantes más cosas que transferir datos. La razón más importante tiene que ver con algunas características del diseño que ya hemos introducido anteriormente, a saber, el procesador interno dual y la cola de instrucciones de estructura tubular (pipeline).

Intel diseñó el 8086/8088 para realizar al mismo tiempo las principales funciones internas de transferencia de datos y búsqueda de instrucciones. Para conseguirlo, tanto el 8086 como el 8088 constan de dos procesadores interconectados en la misma pieza de silicio (tal como se muestra en la figura 3.6). Una unidad está encargada de buscar instrucciones y la otra de ejecutarlas. Además, la unidad encargada de buscar instrucciones utiliza un método llamado *de estructura tubular (pipeline)* o *por cola* para almacenar nuevas instrucciones hasta que se necesiten.

Al procesador principal se le llama unidad de ejecución (EU: Execution Unit). Está encargado de codificar y ejecutar todas las instrucciones. La EU es idéntica en ambos chips, el 8086 y el 8088. Al otro procesador se le llama la Unidad de Interfaz de Bus (BIU: Bus Interface Unit). La BIU está encargada de localizar las

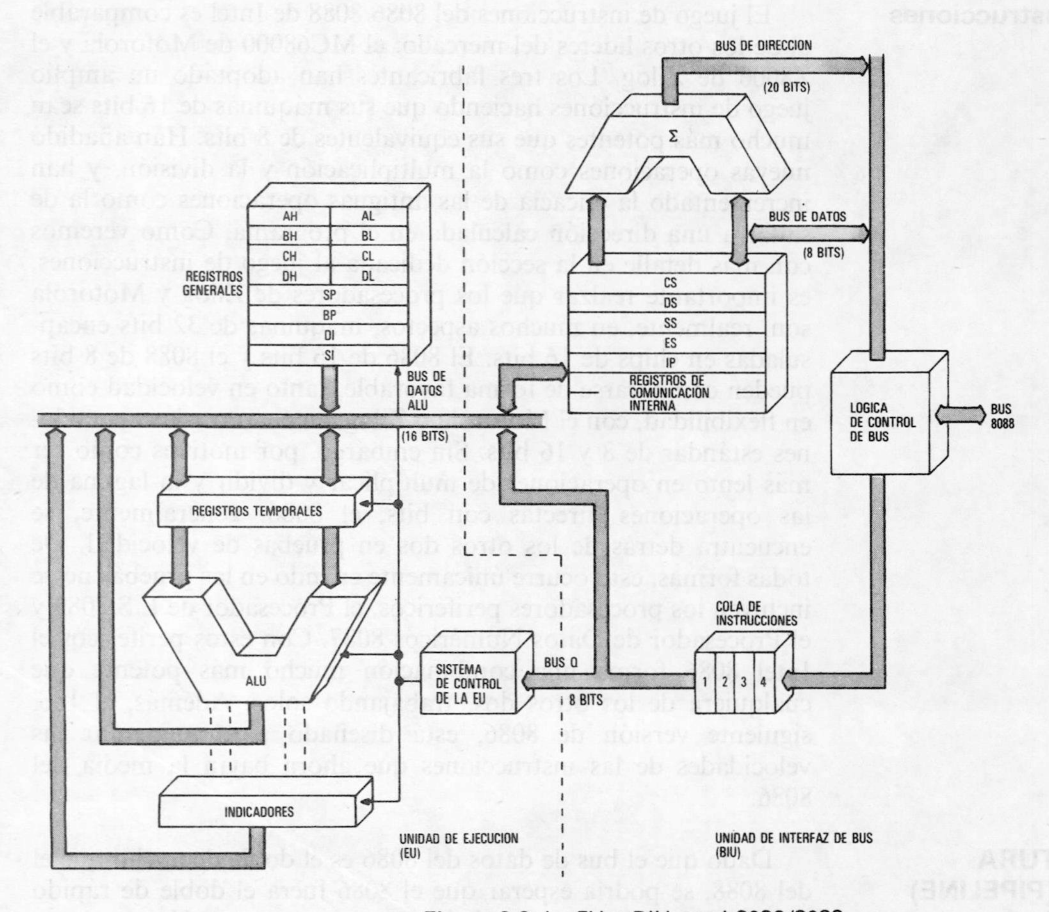


Figura 3.6: La EU y BIU en el 8086/8088.

instrucciones y de transferir todos los datos entre los registros y el mundo exterior. La BIU del 8088 es más compleja, ya que debe transferir datos entre el bus de datos interno de 16 bits y el bus de datos externo de 8 bits.

Cuando la BIU localiza en memoria un octeto de código máquina, lo coloca en una línea de espera especial llamada *cola de instrucciones*[2]. En el 8086 esta cola tiene una longitud de 6 octetos y el código máquina se almacena en memoria de 2 en 2 octetos; en el 8088 la cola de instrucciones tiene sólo 4 octetos de longitud y el código máquina se guarda de octeto en octeto. La división del trabajo entre la EU y la BIU ahorra un tiempo considerable y ayuda a que el rendimiento del 8088 de 8 bits sea más comparable al 8086 de 16 bits.

La situación es muy similar a la de un jefe que tiene una secretaria para coger el teléfono y para la correspondencia. La secretaria trabaja al mismo tiempo que el jefe hace otras cosas, liberándole de muchos detalles del mundo exterior. Las llamadas que se acumulan para el jefe se tratan de una en una, y en el orden que prefiera el jefe.

JUEGO DE REGISTROS

El 8086/8088 contiene 14 registros de 16 bits. Algunos pertenecen a la EU y otros a la BIU. Los de la EU se suele usar para direccionamiento.

Como puede verse en la figura 3.7, la EU tiene los siguientes registros:

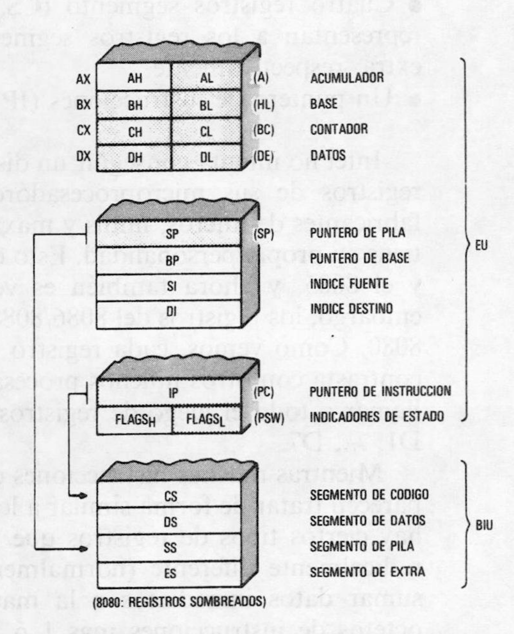


Figura 3.7: Juego de registros del 8086/8088.

---

[2] Una «cola» es una forma de almacenar datos en la cual el primero que entra es el primero que sale —como una cola de un supermercado.

- Cuatro registros generales de 16 bits (AX, BX, CX, DX), que pueden subdividirse (y direccionados separadamente) en ocho registros de 8 bits (AH, AL, BH, BL, CH, CL, DH y DL). En este caso la *X* representa *extendido* (16 bits); la *H, alto,* y la *L, bajo. A* representa *acumulador; B, base; C, contador,* y *D, datos.*
- Cuatro registros punteros y de índice (SP, BP, SI y DI), los cuales no pueden subdividirse. SP es el puntero de pilas, BP es el puntero base y SI y DI indican a los registros índices y fuente, respectivamente.
- Un registro de indicadores de 16 bits, que contiene varios bits de estado para el procesador. Estos incluyen: indicador de cero (ZF), indicador de signo (SF), indicador de paridad (PF), indicador de acarreo (CR), indicador auxiliar (AF), indicador de dirección (DF), indicador de interrupción (IF), indicador de desbordamiento (OF), indicador de desvío (TF).

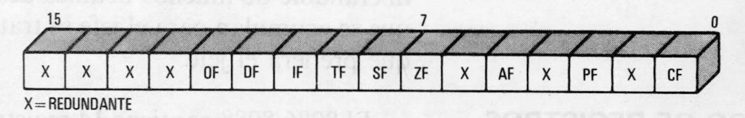


Figura 3.8: Registro de indicadores en el 8086/8088.

La BIU tiene los restantes registros:

- Cuatro registros segmento (CS, DS, SS y ES). Sus códigos representan a los registros segmento de código, datos, pila y extra, respectivamente.
- Un puntero de instrucciones (IP).

Intel no intenta conseguir un diseño simétrico para el juego de registros de sus microprocesadores, tal como lo hacen otros fabricantes de micros, minis y maxi-computadoras. Cada registro tiene su propia personalidad. Esto era cierto para el 8008, el 8080 y el 8085, y ahora también es verdad para el 8086/8088. Sin embargo, los registros del 8086/8088 son más potentes que los del 8080. Como vemos, cada registro tiene su nombre propio. Esto contrasta con otros muchos procesadores en los cuales es normal llamar a todo el juego de registros como R0, R1, ..., R7, o D0, D1, ..., D7.

Mientras muchas instrucciones como ADD, SUB, AND y OR parecen tratar de forma similar a los registros AX, BX, CX y DX, hay ciertos tipos de registros que producen un código máquina radicalmente diferente (normalmente más corto). Por ejemplo, sumar datos inmediatos a la mayoría de registros requiere 2 octetos de instrucciones más 1 ó 2 octetos para almacenar los datos. Sin embargo, sumar un dato inmediato al acumulador (AX) requiere únicamente 1 octeto, más los necesarios para almacenar los datos. Por ello es más eficiente usar AX para cálculos en que intervengan constantes.

Hay también otras instrucciones que usan en los registros. Por ejemplo, la instrucción LOOP usa el registro contador (CX) para almacenar la cuenta de número de iteraciones del bucle. La instrucción de ajuste decimal (DAA) únicamente trabaja con AL, que es la parte más baja del acumulador. Las instrucciones de multiplicar y dividir usan el registro acumulador (AX) y el de datos (DX).

El registro AX es el acumulador de 16 bits. Usándolo a veces se provoca que el ensamblador produzca un lenguaje máquina codificado en muy pocos octetos. Su parte más baja, AL, corresponde al acumulador de 8 bits del 8080.

El registro CX se usa a menudo para almacenar datos, para contar cosas como bucles (para la instrucción LOOP), la iteración de movimientos de cadenas, desplazamientos y rotaciones (justamente CL para estos dos últimos tipos de instrucciones). El registro CX corresponde al registro par BC del 8080.

El registro DX se utiliza para almacenar datos de 16 bits. Puede pensarse que es una extensión del registro AX para multiplicaciones y divisiones con 16 bits. Es una especie de extensión del registro DE del 8080. Sin embargo, el registro DX del 8086 no puede utilizarse para direccionamiento indirecto tal y como lo hace el registro DE del 8080 con las instrucciones LDAX y STAX. Los registros SI y DI del 8086 son los encargados de realizar el direccionamiento indirecto. Esto es motivo de algunos problemas al traducir código del 8080 al 8086.

El registro BX (base de propósito general) se utiliza como registro base para los direccionamientos. El BX es una versión más potente del registro par HL del 8080. Los registros de propósitos específicos como los de índice fuente (S), índice destino (DI) y el puntero base (BP) se utilizan como partes de los modos de direccionamiento. O sea, se utilizan para ayudar en la localización de datos.

Los registros puntero de instrucciones (IP) y puntero de pilas (SP) se encargan del control del flujo propio del programa. En realidad la mayoría de procesadores tiene registros de este tipo.

Los registros segmentos en el BIU tienen funciones especiales que explicaremos más adelante en este mismo capítulo, en la sección de segmentación.

Es interesante comparar esta arquitectura con las del MC68000 y del Z8000. Estos dos procesadores tienen sistemas de numeración uniformes para sus registros. El Motorola MC68000 tiene registros de direcciones y registros de datos. Tales registros tienen 32 bits, y las instrucciones del Motorola los utilizan para guardar datos de 8, 16 y 32 bits. Por otro lado el Zilog Z8000 tiene también un bus de datos de 16 bits, pero sus registros son de 16 bits. Para guardar datos de 32 bits el Z8000 debe agrupar sus registros por pares, construyendo un registro de 32 bits a partir de dos de 16. Puede incluso formar registros de 64 bits agrupando cuatro registros de 16 bits. Esto es similar a la forma en la que los procesadores Intel 8080, 8085, 8086 y 8088 forman un registro de 16 bits a partir de dos de 8 bits. En general el Z8000 y el 68000

tienen más registros y casi todos sus registros de datos admiten un uso general.

SEÑALES Y TERMINALES

Esta sección no es necesaria para aquellos interesados únicamente en la programación del 8086/8088. Contiene una descripción de las distintas señales eléctricas que aparecen sobre los terminales de chip 8086/8088. Si está interesado en el tema, vale la pena que antes revise de nuevo la sección dedicada a la estructura de interrupciones del chip. Hay algunos conceptos software importantes en ella.

Modo máximo y modo mínimo

El 8086/8088 puede conectarse al circuito de dos formas distintas: el modo máximo y el modo mínimo. El modo queda determinado al colocar un determinado terminal (llamado MN/MX) a tierra o a la tensión de alimentación. El 8086/8088 debe estar en modo máximo si quiere trabajar en colaboración con el Procesador de Datos Numérico 8087 y el Procesador de Entrada/Salida 8089. En los dos modelos de computadora vistos en el capítulo 2 (el primero utiliza un 8086 y el segundo un 8088), la CPU estaba conectada en modo máximo debido a que ambos diseños utilizaban el NDP 8087 y el IOP 8089 formando un pequeño *grupo* junto a la CPU.

En el modo máximo, el 8086/8088 depende de otros chips adicionales como el Controlador de Bus 8288 para generar el conjunto completo de señales de control de bus. El modo mínimo permite al 8086/8088 trabajar de una forma más autónoma. La figura siguiente muestra un esquema de ambos modos.

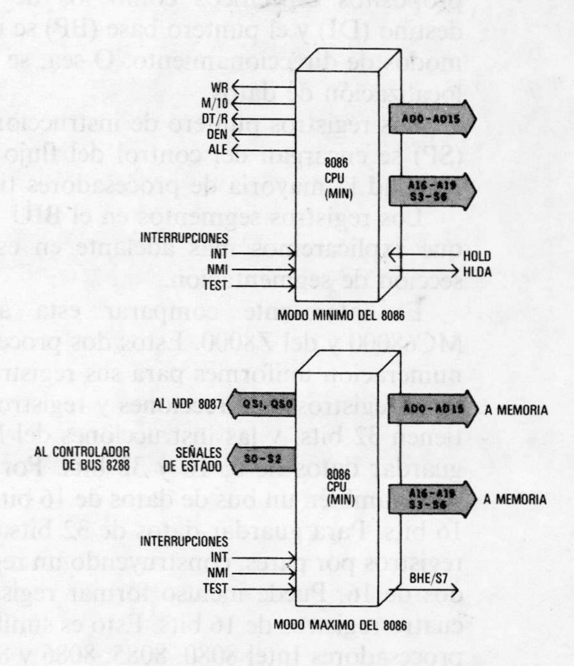


Figura 3.9: Modos mínimo y máximo del 8086.

En ambos modos, las señales del 8086/8088 (véase la figura 3.1 para los terminales) se pueden agrupar de la siguiente manera:

Alimentación
Reloj
Control y estado
Direcciones
Datos

Hay tres terminales para la alimentación: tierra (GND) en los terminales 1 y 20, y una tensión de entrada de 5 voltios (Vcc) en el terminal 40. La tierra es tierra a la vez para la alimentación y para las señales.

Hay una señal de reloj en el terminal 19.

Tanto para el 8086 como para el 8088 hay 20 bits de dirección. Los 4 bits más significativos de la dirección comparten terminales con algunas de las señales de estado. En el 8086, los *16 bits menos significativos* se comparten con los bits de datos; y en el 8088 sólo los *8 bits menos significativos* comparten terminales con los datos, debido, desde luego, a que el bus externo de datos es de 16 bits para el 8086 y de 8 bits para el 8088. En ciertos instantes, tales terminales conducen parte de una dirección, y en otros llevan información sobre el estado y los datos. El Latch (memoria) 8282 está diseñado para seleccionar la información sobre la dirección de dichos terminales en el instante preciso, e ignorar el estado y los datos.

Hay varios grupos de control y señales de estado.

El terminal MN/MX controla si el procesador está en modo mínimo o máximo, conectándolo a tierra o a una tensión de 5 voltios.

Del S0 al S7 son señales de estado en los terminales 26, 27, 28, 38, 37, 36, 35, 34, respectivamente (note el ligero «desplazamiento» de estas asignaciones). En ciertos momentos son salidas del procesador. En otros momentos, aparecen otras señales distintas en los mismos terminales. Mirando el estado pueden decirse cosas tales como el tipo de acceso al bus (lectura o escritura, memoria o E/S), el registro de segmento en uso y el estado del sistema de interruptores. S0, S1, S2 son sólo accesibles en modo máximo, en cuyo caso se introducen en los chips controladores de bus 8288. El controlador de bus 8288 genera a partir de estas otras importantes señales de control. En el modo mínimo el 8086/8088 produce alguna de las señales del 8288 directamente.

RD es una señal de estado generada por el procesador sobre el terminal 32. Indica cuando el procesador lee (sea de memoria o de E/S).

La señal READY (terminal 22) es una entrada *de* los dispositivos externos (memoria o controladores E/S).

La señal READY pasa a través del generador de pulsos 8284 para sincronizarse con la señal de reloj.

La figura 3.10 muestra un diagrama de cómo se realiza esto. La señal READY trabaja de la siguiente forma: Si se ha seleccio-

nado un dispositivo externo para lectura o escritura y todavía no está preparado para completar la transferencia de datos, pone a cero la línea de señal READY. El procesador ve esta señal y añade ciclos extras de «espera» hasta que la línea READY vuelve a su nivel normal de 5 voltios indicando que el dispositivo externo está preparado para realizar la transferencia. Acabada la transferencia las actividades del procesador continúan normalmente.

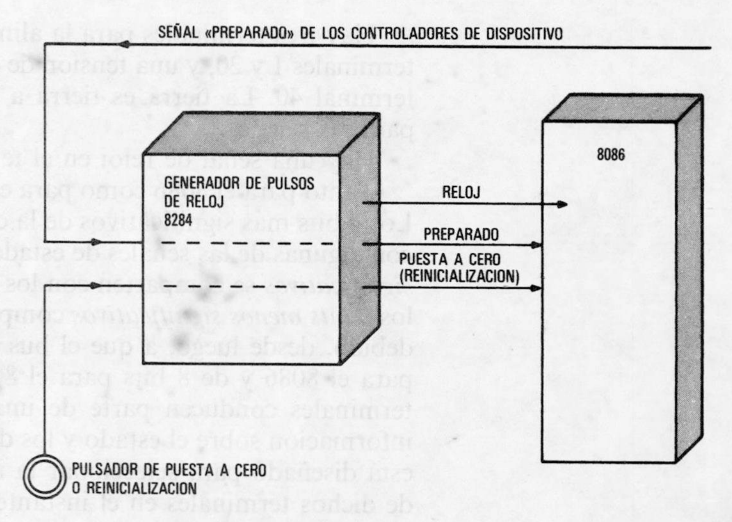


Figura 3.10: Señales que pasan a través del generador de pulsos de reloj 8284.

La señal RESET (terminal 21) es otra de las entradas que también pasa por el generador de pulsos 8284 para sincronizarse con la señal de reloj. RESET se utiliza para inicializar el procesador; lo mismo que si se hubiese desconectado y vuelto a enchufar. Resulta bastante más sencillo reinicializar el sistema con la señal RESET que no desactivar y volver a activar cada una de las componentes eléctricas del circuito. La señal RESET borra la cola de instrucciones y ciertos registros (los indicadores; puntero de instrucción, y segmento de datos, de pila y registros de segmentación adicionales). Tras un RESET, el registro de segmento de código se pone al valor FFFFH. Veremos más adelante que con esto se fuerza al procesador a leer el primer octeto de instrucción en la dirección FFFF0H.

Los terminales NMI (Non-Maskable Interrupt: Interrupción no enmascarada —terminal 17) e INTR (INTeRrupt Request: Petición de interrupción —terminal 18) son parte del sistema de interrupciones del 8086/8088. Un pulso en la línea NMI provoca una interrupción especial llamada *interrupción tipo 2*. Una señal en la línea INTR causará una respuesta de interrupción de tipo general. El término «no-enmascarable» se refiere al hecho de que la interrupción generada por el NMI no se puede activar o desactivar vía un comando software a la CPU. Las interrupciones generales por INTR pueden desactivarse vía software. (Ver la

sección de interrupciones del 8086/8088 donde se explica más detalladamente este tema.) En el modo mínimo salen las siguientes señales de control y de estado: SS0 (Estado S0 —terminal 34), HLDA (terminal 30), WR (control de escritura —terminal 29), M/IO (control de memoria de E/S —terminal 28), DT/R (Recepción/Transmisión de datos —terminal 27), DEN (Datos Accesibles —terminal 26), ALE (Latch de direcciones activo —terminal 25), INTA (Reconocimiento de Interrupciones —terminal 24). Una de las señales es de entrada; HOLD (terminal 31). La señal M/IO del 8088 se invierte para hacerla compatible con las señales del 8085, y la línea SS0 no existe en el 8086 (el terminal correspondiente se utiliza para la señal BHE/S7). La señal BHE (terminal 34) se usa en el direccionamiento de 16 bits. Veamos todas estas señales con más detalle.

El terminal SS0 (estado S0) del 8088 se utiliza junto a las señales de control de Memoria/ES (M/IO) y la Recepción/Transmisión de Datos (DT/R) para determinar el estado del 8088, es decir, para conocer exactamente en qué estado está (obtención de código, escritura en memoria, lectura de memoria, escritura en port de E/S, lectura de un port de E/S, etc.).

En el 8086, el BHE se utiliza como ayuda en la interfaz de los dispositivos de 8 bits con el bus de datos de 16 bits. Si la línea de dirección 0 es 0 (indicando una dirección par), la señal BHE especifica si se está direccionando una palabra entera o un solo octeto. BHE igual a 0 significa que se trata de una palabra, BHE igual a 1 significa que es un octeto. Si la línea de dirección 0 es 1 (indicando una dirección impar), el 8086 direcciona siempre un octeto y no una palabra. En este caso la señal BHE es 0.

Las señales de datos accesibles (DEN) y recepción/transmisión de datos (DT/R) se usan para especificar el transceptor de bus 8286 donde hay datos y la dirección de destino (entrada o salida). La señal ALE dice al Latch de Dirección Octal 8282 donde encontrar la dirección.

La señal M/50 dice al sistema cuando el 8086/8088 requiere acceso a la memoria o al espacio de E/S. La señal WR de selección de escritura indica que los datos están disponibles en las líneas de datos. Los aparatos externos (tanto memoria como controladores de E/S) pueden usar esta señal para ayudar a cargar los datos en sus registros. En el modo máximo hay señales de selección de escritura separadas para E/S y para memoria. (Son generadas por el controlador de bus 8288.)

Las señales HOLD, HLDA son parte del propio sistema de control de bus del 8086/8088. Cuando otro procesador o un aparato como un controlador DMA quiere acceder al control del bus, manda una señal al 8086/8088 a través de la línea HOLD. Cuando está preparado para hacerlo, el 8086/8088 pone sus líneas de datos/direcciones y la mayoría de líneas de control en el estado de alta impedancia (desconectado eléctricamente del bus). Al mismo tiempo, el 8086/8088 envía la señal HLDA para indicar que el bus está libre. El otro aparato puede usar ahora el bus. Cuando finaliza con el bus, envía una señal en la línea HOLD.

Inmediatamente después de recibir la señal, el 8086/8088 reanuda el uso del bus.

En el modo máximo, las siguientes señales son de salida: S2, S1, S0 (estado —terminales 28-26); QS1, QS0 (estado de la cola —terminales 25 y 24); RQ/GT1, RQ/GT0 (Petición/concesión —terminales 31 y 30), y LOCK (terminal 29). Realmente, las líneas de petición/concesión (RQ/GT) también pueden aceptar señales de entrada. Ahora lo veremos más detalladamente.

Las dos señales de petición/concesión (RQ/GT) y la señal LOCK se utilizan para coordinar las actividades con otros procesadores en una configuración de *grupo*. Un *grupo* es un subsistema que consta de uno o más procesadores y buses que los interconectan. El *grupo* se conecta a la computadora a través del *bus de sistema* (bus principal) vía el chip de interfase de bus, el cual se explicará en el capítulo de chips soportes. Las señales RQ/GT operan de la misma forma que lo hacen la HOLD y HLDA para transferir el control del bus (tal como se describe anteriormente).

La señal LOCK comunica a otro procesador el sistema cuando *no* debe intentar tomar el control del sistema. Esto es importante ya que permite que los esquemas de protección, definidos vía software puedan operar correctamente. Estos esquemas de protección requieren que un bloque de código al cual acceder dos o más procesadores esté protegido de forma que un procesador no lo pueda cambiar, mientras el otro lo está leyendo para realizar otro trabajo. Véase la sección del capítulo 2 sobre multi-proceso y la explicación del octeto de ocupación del capítulo 5, para mayor detalle sobre este tema.

QS1 y QS0 son señales de estado de las colas de instrucciones. Sólo son utilizables en modo máximo. El procesador de datos numérico 8087 utiliza estas señales para coordinarse con el 8086/8088.

La señal TEST (terminal 23) se utiliza para enlazar el 8086/8088 con un procesador paralelo, tal como el procesador numérico 8087, sincronizando el procesador principal con los otros. Describiremos esto, con más detalle, en el capítulo 4.

Una *estructura de interrupción* es una forma de que el procesador provea un servicio rápido y uniforme para la E/S, correcciones y ciertos tipos de error. En general, el procesador continúa con su trabajo habitual hasta que ocurre una interrupción, en cuyo momento salva su estado actual (puntero de instrucción, segmento código e indicadores); ejecuta una rutina especial, y entonces vuelve a lo que estaba haciendo antes. Se puede ver la interrupción como una llamada a una subrutina y la rutina especial de interrupción como el cuerpo de la subrutina. Las principales diferencias entre subrutinas e interrupciones son: 1) las subrutinas son llamadas únicamente por instrucciones software, mientras que las interrupciones pueden ser invocadas tanto por software o hardware; 2) las subrutinas únicamente deben salvar la dirección de retorno, mientras que las interrupciones guardan dicha dirección y el estado de todos los indicadores, y

3) las subrutinas necesitan tener un medio de pasar datos al programa principal y viceversa, mientras que esto no es necesario para las interrupciones.

Varios aparatos pueden generar interrupciones hardware, como por ejemplo, el NDP 8087, el IOP 8089, y el controlador de interrupciones programable 8259. ¡El 8086/8088 puede, él solo, generar interrupciones! Detalle de cómo y cuándo pueden otros aparatos generar interrupciones se encuentra en las secciones relativas a cada aparato en el resto de este capítulo. La interrupción (INT) 8086/8088 puede generar interrupciones software.

La estructura de interrupciones del 8086/8088 utiliza una tabla de 256 posiciones de 4 octetos cada una, la cual está en el inicio absoluto de la memoria. Cada una de estas posiciones de la tabla de interrupciones puede cargarse con un puntero a diferentes rutinas de la memoria principal. Estos punteros contienen el nuevo contenido del segmento código (2 octetos) y el puntero de instrucciones (2 octetos) para la rutina que puede estar localizada en cualquier parte de la memoria. En la sección de segmentación explicaremos por qué se necesitan ambas cantidades. Mostraremos cómo se calcula a partir de estas dos cantidades de dirección real de memoria. A cada uno de estos punteros de 4 octetos se le asigna un número del 0 al 255, según su posición en la memoria. A este número se le llama *tipo*. Al tipo de interrupción 0 se le asigna la posición de memoria 0, al tipo 1 se le asigna la posición 4, y así hasta la posición 1.020. En general, al tipo *n* se le asigna la posición *4n* de memoria. No tienen por qué cargarse todos los punteros, pero antes de que se use una interrupción de cierto tipo, el correspondiente puntero de la tabla de interrupciones deberá cargarse para su rutina de servicio (véase la figura 3.11).

Cada tipo de interrupción puede ser llamado tanto por hardware como por software. Esto hace posible probar vía software las interrupciones hardware. Hay un bit de control, llamado indicador de interrupción (IF) que controla si el 8086/8088 responde o no a las interrupciones externas. Este bit puede activarse o desactivarse por medio de las órdenes de activar interrupciones (STI) y el de borrar interrupción (CLI). Esto abre y cierra la puerta para interrupciones de la CPU. Cuando el bit de interrupción está desactivado es como si hubiera un cartel de «no molestar» en dicha puerta.

Las interrupciones hardware actúan de la siguiente forma: Cuando un dispositivo externo necesita servicio, produce una señal en la línea de petición de interrupción (INTR) del 8086/8088. Si el 8086/8088 puede responder (interrupción activa), envía una señal de «recibido», bien de forma directa (en modo mínimo), o bien vía el controlador de bus 8288 (en modo máximo).

El dispositivo externo indica, mediante 1 octeto en el bus de datos, qué tipo de interrupción desea. El 8086/8088 usa este número para localizar el puntero de la tabla de interrupciones. A continuación, el 8086/8088 salva las condiciones de sus indicadores y un puntero en la pila con la dirección de retorno, y carga el

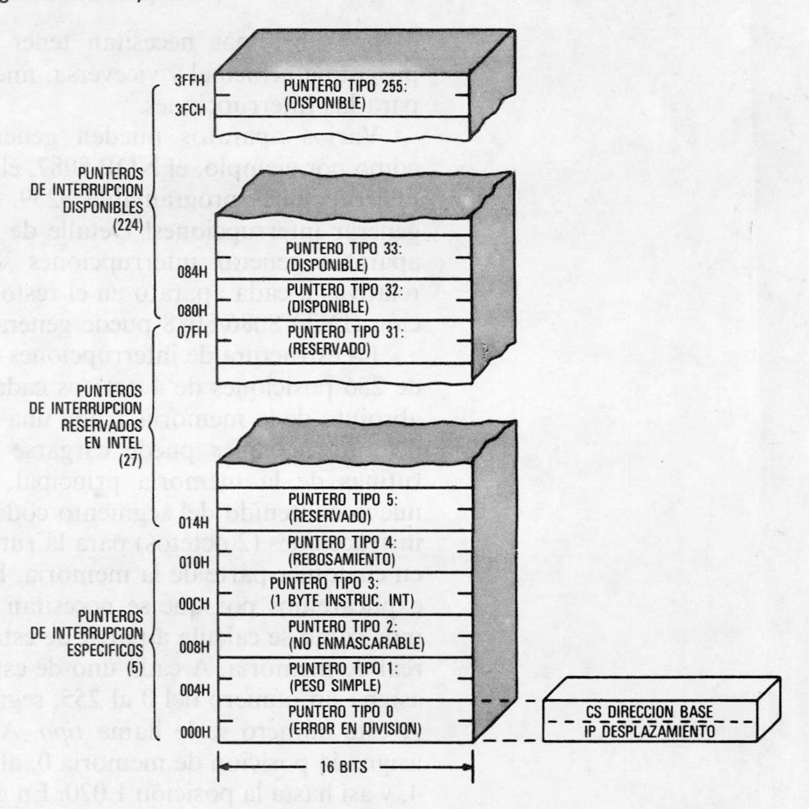


Figura 3.11: Tabla de punteros de interrupción en el 8086/8088.

segmento código y el puntero de instrucciones desde la tabla de interrupción. Esto hace que el procesador ejecute la rutina de servicio. Al final de dicha rutina debe haber una instrucción de retorno de interrupción (IRET). Esta interrupción restaura los indicadores, el segmento código y el puntero a la dirección de retorno. Se debe usar la instrucción IRET ya que una instrucción retorno «normal» no restauraría los indicadores, por lo que no sería adecuada para estos propósitos (de otra manera se rompería la pila). En este punto, el procesador ha vuelto a sus operaciones normales.

Hay una clase especial de interrupciones llamadas *interrupciones no enmascarables*. El término «no enmascarable» se refiere al hecho de que esta clase de interrupciones no pueden desactivarse limpiando el indicador de interrupciones. Esta clase de interrupciones genera una interrupción del tipo 2; o sea, su puntero se encuentra en la dirección $2 * 4 = 8$ de memoria. Las interrupciones no enmascarables se reservan para casos de emergencia como fallos de potencia o errores de memoria.

Tipos especiales de interrupciones se generan al dividir por cero o en capacidades excedidas.

También hay tipos especiales que se usan para pruebas de programas. La interrupción de paso simple ocurre después de

cada instrucción si el indicador de interrupción está activo. Desafortunadamente, el procedimiento para activar y desactivar dicho indicador es completo. Implica poner los indicadores en la pila, cambiar su *imagen* y devolverlo a su sitio. (Por imagen indicamos una *copia* de los indicadores tal como están en el registro de indicadores, pero no los bits indicadores reales.) La interrupción de paso simple causa una interrupción del tipo 1. Si se coloca un puntero a una rutina en la posición $1*4=4$, que muestra el contenido de los registros, se puede ver el progreso del procesador paso a paso. Otro tipo de interrupción útil para pruebas es la interrupción por punto de acceso. Esta es una forma especial de interrupción software en la cual el código máquina tiene sólo 1 octeto en vez de los 2 usuales. Esto hace que sea fácil llevarlo a su sitio. Genera una interrupción de tipo 3. Esta clase de interrupción se usa de la forma siguiente: El programador selecciona una dirección donde desea detener el procesador y examinar el contenido de los registros e indicadores. El programador inserta el tamaño de octeto código máquina para esta interrupción en la dirección seleccionada y ejecuta el programa. Cuando el procesador llega a este punto, salta a la rutina de servicio de tipo 3. Esta rutina debe devolver el control al programador para que haga lo que quiera, quizás mostrar el contenido de los registros, quizás un volcado de memoria u otra cosa cualquiera. El capítulo 7 incluye un ejemplo de esto.

MODOS DE DIRECCIONAMIENTO

El 8086/8088 tiene 25 modos diferentes de direccionamiento (véase tabla 3.1) o reglas para localizar un operando de una instrucción. Dichos modos son algo complicados, pero pueden verse como casos especiales de los casos tipo: referencia a registros y referencia a memoria. En el primer caso, el operando está localizado en un registro específico. Sin embargo, deben sumarse cuatro cantidades para obtener la dirección de un operando en memoria. Dichas cantidades son: 1) *dirección de segmento,* 2) *dirección base,* 3) una cantidad *índice* y 4) un *decalage*. Véase la figura 3.12 que muestra un caso general de direccionamiento a memoria.

La dirección del segmento se almacena en el *registro de segmentación* (DS, ES, SS o CS); en la próxima sección explicaremos la forma en que se hace esto. Por ahora, basta con saber que el contenido del registro de segmentación se multiplica por 16 (desplazado cuatro dígitos binarios a la derecha), antes de utilizarse para obtener la dirección real. El registro de segmentación *siempre* se usa para referenciar a memoria.

La *base* se almacena en el *registro base* (BX o BP). El índice se almacena en el *registro índice* (SI o DI). Cualquiera de estas cantidades, las dos, o ninguna, pueden utilizarse para calcular la dirección real. El programador puede usar tanto la base como el índice de diferentes formas para gestionar ciertas cosas, tal como matrices de dos dimensiones, o estructuras internas a otra estructura, esquemas que se utilizan en las prácticas modernas de programación.

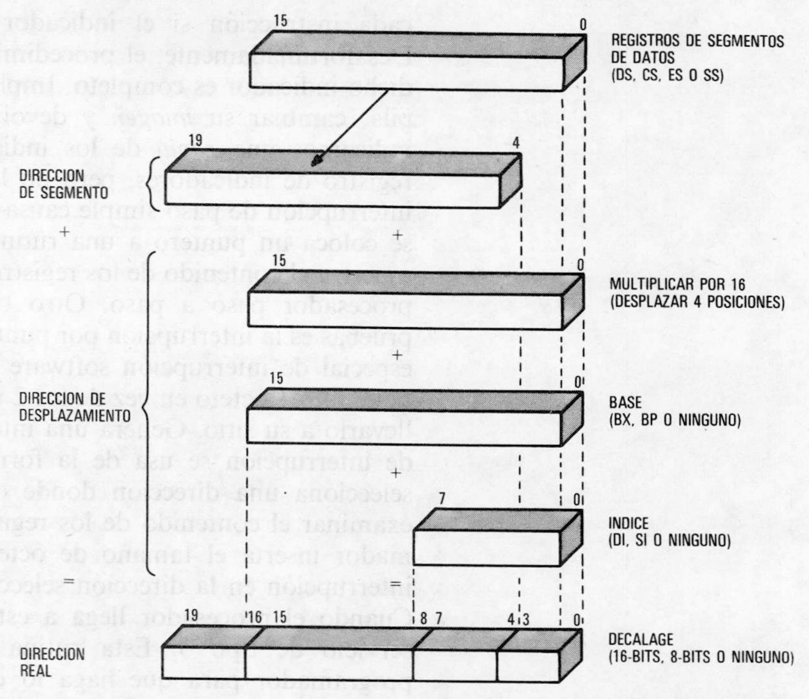


Figura 3.12: Gráfico de modos de direccionamiento en el 8086.

La base y el índice son variables o dinámicas, ya que están almacenadas en registros de la CPU de propósito general. Es decir, pueden modificarse fácilmente mientras se ejecuta un programa. Los nombres de estas cantidades dan a entender que la base se modificará menos que el índice; pero esto queda realmente a criterio del programador.

Además del segmento, base e índice se usan unos *decalages* (desplazamientos) de 16 bits, 8 bits o 0 bits (inexistente). Este decalage es una cantidad *estática*. Se fija el tiempo de ensamblaje (paso de código fuente a código máquina) y no puede cambiarse durante la ejecución del programa (ni debe cambiarse).

El decalage se utiliza para cosas como compilar datos, organizar la memoria y reubicar más rápida y fácilmente. Por ejemplo, imaginemos que deseamos acceder a un octeto variable que se encuentra siempre en la quinta posición de una tabla de 40 octetos, y supongamos que esta tabla se carga de memoria en distintos momentos, cada vez en una posición distinta (dependiendo de la que haya en memoria en ese momento); para acceder a esta variable usted deseará utilizar un modo de direccionamiento con una *base* y un decalage de 8 bits.

La base será igual a la dirección de comienzo en curso de la tabla y el decalage de 8 bits siempre será 5. Como la base se guarda en un registro base (normalmente el BX), se puede cambiar fácilmente cuando cambie la posición de la tabla. Además, como el 5 es una constante, se puede almacenar como decalage estático.

En el lenguaje ensamblador, ciertos caracteres separadores colocados alrededor del operando, como paréntesis o corchetes, indican el modo de direccionamiento. Todo esto se traduce al lenguaje máquina de formas más o menos complejas, dependiendo de la instrucción y del modo de direccionamiento elegido.

mm	aaa	Parte de desplazamiento de la dirección
00	000	(BX) + (SI)
00	001	(BX) + (DI)
00	010	(BP) + (SI)
00	011	(BP) + (DI)
00	100	(SI)
00	101	(DI)
00	110	Dirección directa
00	111	(BX)
01	000	(BX) + (SI) + número 8 bits
01	001	(BX) + (DI) + número 8 bits
01	010	(BP) + (SI) + número 8 bits
01	011	(BP) + (DI) + número 8 bits
01	100	(SI) + número 8 bits
01	101	(DI) + número 8 bits
01	110	(BP) + número 8 bits
01	111	(BX) + número 8 bits
10	000	(BX) + (SI) + número 16 bits
10	001	(BX) + (DI) + número 16 bits
10	010	(BP) + (SI) + número 16 bits
10	011	(BP) + (DI) + número 16 bits
10	100	(SI) + número 16 bits
10	101	(DI) + número 16 bits
10	110	(BP) + número 16 bits
10	111	(BX) + número 16 bits
11	000	registro AX (palabra) o AL (octeto)
11	001	registro CX (palabra) o CL (octeto)
11	010	registro DX (palabra) o DL (octeto)
11	011	registro BX (palabra) o BL (octeto)
11	100	registro SP (palabra) o AH (octeto)
11	101	registro BP (palabra) o CH (octeto)
11	110	registro SI (palabra) o DH (octeto)
11	111	registro DI (palabra) o BH (octeto)

Nota: mm significa los 2 primeros bits del octeto de direccionamiento y aaa los 3 últimos bits de dicho octeto.

Tabla 3.1: Modos de direccionamiento del 8086/8088.

El siguiente ejemplo da una visión esquemática de la situación general. En este caso la instrucción es la instrucción INC (incrementar) y el único operando está en memoria. Se accede a él con un modo de direccionamiento que usa la base, el índice y un decalage de 8 bits. En nuestro caso, la dirección del segmento está

en el segmento de datos, la base está en el registro base (BX), el índice está en el registro DI, y el decalage es 6.

En lenguaje ensamblador, la instrucción aparecería así:

```
INC     6[BX][DI]          ; Incrementa
                           ; Contenido
                           ; de BX+DI+6
```

Obsérvese que el comentario se refiere únicamente a la dirección relativa en el segmento. Hay dos razones para ello. La primera es que no hay mucho espacio para poner comentarios, y la segunda es que el programador únicamente debe pensar en términos relativos al segmento, ya que los diversos segmentos se determinan por medio de los registros de segmentación.

Veamos ahora esta referencia en términos totalmente numéricos. Supongamos que BX contiene 4.000 h (h=hexadecimal); DI contiene 20 h y DS contiene 3.000 h. El desplazamiento se determina de la siguiente forma:

desplazamiento=decalage+(BX)+(DI)=6+4.000+20=4.026 h

Como hemos visto en la sección de segmentación, el contenido del registro de segmentación (en este caso, registro de segmento de datos) se multiplica por 16 (decimal) antes de añadírsele el resto. Dado que 16 en decimal es 10 en hexadecimal, esto produce un desplazamiento de un dígito hexadecimal (4 bits) a la izquierda. Combinando esto con la dirección relativa en el segmento, nos da:

Dirección real = 10 * (DS)+despl.=3.000 * 10 + despl.
=30.000+4.026
=34.026 h

Así, 34.026 h es la dirección real de memoria donde encontraríamos este operando, mientras 4.026 es la dirección en el segmento, que es lo que interesa al programador.

Como hemos visto, este esquema de modo de direccionamiento, con su dúo de cantidades estáticas y dinámicas, se conjuga bien con las técnicas modernas de programación modular o la programación tipo *caja-dentro-de-caja*. El registro BX es una constante dentro de la caja exterior (módulo principal del programa), el registro DI es constante en la caja interior (último modelo de un programa), y el decalage apunta a una dirección particular en la caja interior. Esto facilita el acceso a estructuras de datos de tipo *caja-dentro-de caja,* tal como matrices de registros, o registros de matrices, o cualquier otra estructura que un lenguaje moderno como Pascal puede soportar o demandar. (Ver *Pascal Primer*, por David Fox y Mitch Waite, Howard Sams.)

Estos modos de direccionamiento producen algunos inconvenientes en el 8086/8088. La CPU gasta tiempo calculando una dirección compuesta de varias cantidades. Principalmente, esto se

debe al hecho de que el cálculo de direcciones en el 8086/8088 está programado en microcódigo. En una futura versión del 8086, la iAPX 186, estos cálculos estarán *cableados* en la máquina y, por lo tanto, costará mucho menos tiempo el realizarlos.

ESTRUCTURA DE MEMORIA DE SEGMENTACION

Como se ha mencionado anteriormente, el 8086/8088 usa un esquema ingenioso llamado segmentación, para acceder correctamente a un megaocteto completo de memoria, con referencias de direcciones de sólo 16 bits.

Veamos cómo funciona. Cualquier dirección en el 8086/8088 tiene dos partes, cada una de las cuales es una cantidad de 16 bits. Una parte es el *desplazamiento* y la otra la *dirección de segmento*. El desplazamiento de 16 bits se compone de varias partes: un decalage (un número fijo), una base (almacenada en el registro base) y un índice (almacenado en el registro índice). La dirección del segmento se almacena en uno de los cuatro registros segmento (CS, DS, ES, SS). El procesador usa estas dos cantidades de 16 bits para calcular la dirección real de 20 bits, según la siguiente fórmula:

Dirección real = 16 * (dirección del segmento) + desplazamiento

Tal como veíamos antes, dado que 16 en decimal es 10 en hexadecimal, multiplicar por 16 decimal es lo mismo que correr una posición a la izquierda un número hexadecimal. Por ejemplo, si la dirección del segmento es 500 h y la del desplazamiento es 234 h, la dirección real será 5.234 h.

Veamos con más detalle la dirección del segmento, que está almacenada en uno de los cuatro registros de 16 bits llamados registros de segmentación.

Los registros de segmentación reciben los nombres siguientes: de código, datos, pila y extra (CS, DS, SS, ES, respectivamente). Las instrucciones se gestionan usando el segmento código, las operaciones con pilas usan el segmento de pila, y para los datos se utilizan el segmento de datos y el extra.

Por ejemplo, supongamos que el procesador ejecuta un programa y el puntero de instrucciones (IP) contiene 234 h y el registro de código contiene 800 h. La siguiente fórmula indica que el siguiente octeto de instrucción se encuentra en la dirección

8.234 = 800 * 10 + 234 (hexadecimal)

de memoria.

Esta dirección de segmento únicamente indica dónde se inicia un segmento. Pero, lo que no hace de ninguna forma es dividir el segmento en *párrafos*.

Hay un octeto especial de *prefijo* que permite *ignorar* algunas de estas asignaciones, pero *no* todas. En particular, si la CPU usa la instrucción puntero para ayudarse a apuntar a memoria (es decir, para localizar parte de una instrucción), entonces el registro código *siempre* se usa. Y si la CPU usa el puntero de pila para

ayudarse a apuntar a memoria (tanto para introducir como para extraer de la pila), entonces se usará *siempre* el registro pila. El caso del destino en operaciones con cadena es el único en el cual hay una restricción. La combinación del índice destino (DI) y el segmento extra (ES) se usa *siempre* para calcular la dirección del destino en cualquier operación con cadenas. El octeto prefijo puede usarse, sin embargo, para obligar a utilizar *alguno* de los cuatro registros segmento en el cálculo de la dirección puente de una operación de cadenas. El valor por defecto (o sea, el valor sin octeto prefijo) es el registro de datos.

Los 24 modos de referenciar la dirección (usados para el acceso de datos) pueden aceptar un octeto prefijado e ignorar sus asignaciones por defecto al segmento, usando *cualquier* registro segmento. La asignación por defecto es o el segmento de datos (DS) o el de pila (SS) y, de hecho, se usa siempre el segmento de datos *a menos* que el modo de direccionamientos use el puntero base (BP), en cuyo caso se usa el registro de pila. Intel aconseja utilizar el puntero base para acceder a datos en la pila del sistema y normalmente con llamadas a subrutina. Por ejemplo, antes de llamar a una subrutina, se pueden introducir varias, digamos *n*, palabras de datos de 16 bits en la pila para que las use la subrutina en sus cálculos. Al principio de la subrutina, se copiará el contenido del puntero de la pila en el puntero base, y en la subrutina usará el BP (con un decalage) para acceder a cualquiera de estas palabras de datos. Véase la figura 3.13 para tener una idea gráfica de este funcionamiento. Se puede también devolver datos al programa principal desde la subrutina de forma similar.

El uso de estos diferentes segmentos significa que hay *áreas de trabajo* separadas para el programa, la pila y los datos. Cada área de trabajo tiene un tamaño máximo de 64K octetos y mínimo de 0. Dado que hay cuatro registros de segmentación, uno de programa (CS), uno de pila (SS) y dos de datos —segmento de

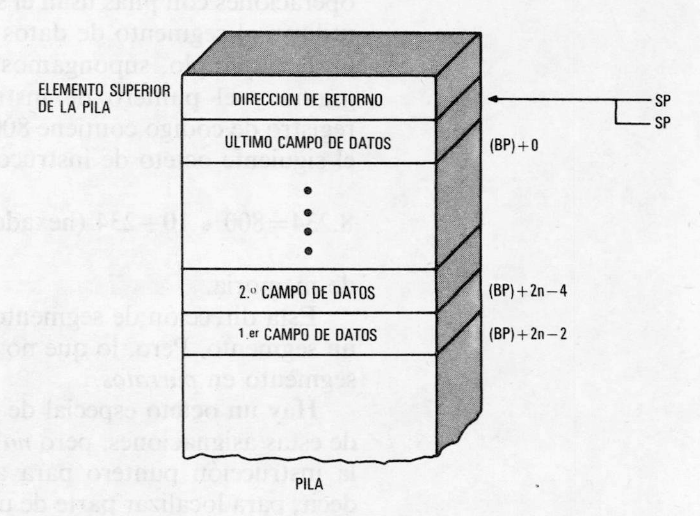


Figura 3.13: Uso del puntero base para acceder a datos en la pila.

datos (DS) y segmento extra (ES)— el área de trabajo puede llegar hasta 4 * 64 K = 256 K en un momento dado. Se supone que las distintas áreas no se superponen. Sin embargo, aunque no es necesario, es posible y aconsejable colocar los cuatro segmentos en la misma área. En este caso, programa, pila y datos residirían en la misma área de trabajo de 64 K.

El programador puede determinar la posición de estos segmentos, cargando el apropiado registro de segmentación de 16 bits con la dirección de segmento apropiada. Esto se realiza normalmente al inicio del programa, pero se puede fácilmente hacer en cualquier momento mientras el programa se ejecuta, cambiando *dinámicamente* la dirección de estos segmentos.

Cambiando el desplazamiento, el programador puede acceder a cualquier punto del segmento. Piense en el segmento como una ampliación de memoria que forma un área de trabajo. El desplazamiento es la única parte de la dirección que aparece normalmente en los programas en lenguaje ensamblador durante las referencias a direcciones del área de trabajo. En la sección de modos de direccionamiento vimos cómo se calculaba dicho desplazamiento usando tres cantidades (base, índice y decalage).

Hemos visto que la dirección del segmento puede ser cualquier número múltiplo de 16. Esto es particularmente sencillo en aritmética hexadecimal. Por ejemplo, en notación hexadecimal las direcciones iniciales de un segmento son 0 h, 10 h, 20 h, 30 h, hasta FFFF0 h. Obsérvese que estas direcciones ocurren en intervalos cerrados razonables y recorren toda la memoria hasta FFF0 h, cubriendo 1 megaocteto fácil y completamente con una perspectiva de 64 K. Obsérvese que el contenido del registro

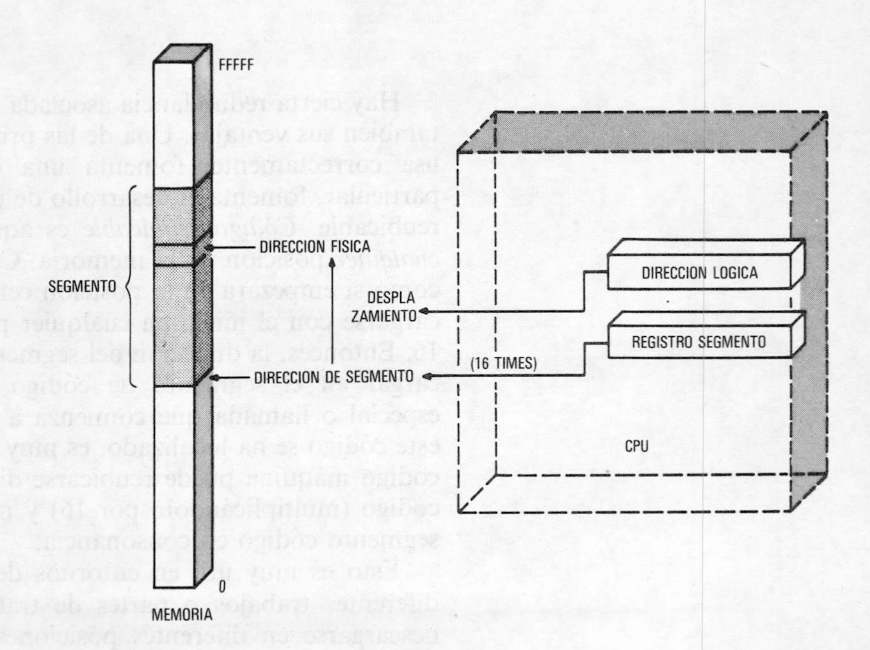


Figura 3.14: Un segmento.

segmento está desplazado un dígito hexadecimal o cuatro binarios a la izquierda, antes de sumarle el desplazamiento. La figura 3.15 nos muestra una visión gráfica de esto.

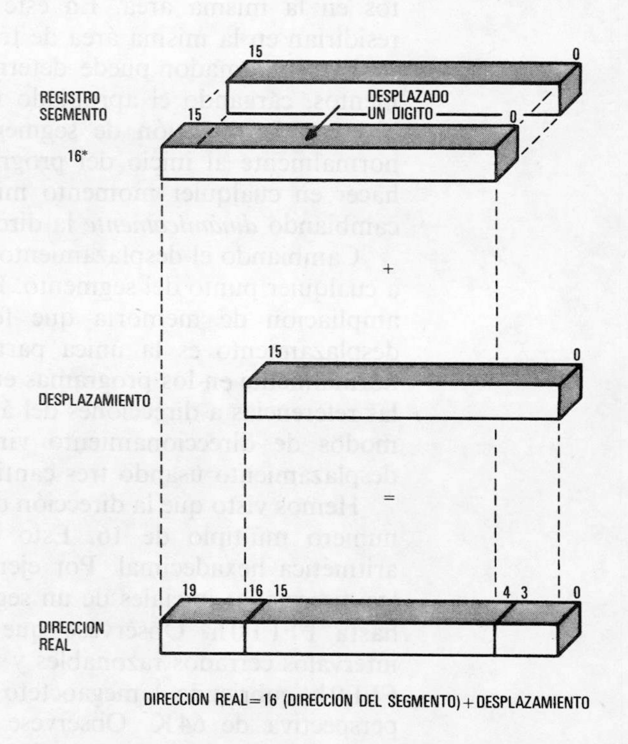


Figura 3.15: Cálculo de la dirección.

Hay cierta redundancia asociada con este esquema, pero tiene también sus ventajas. Una de las principales ventajas es que si se usa correctamente, fomenta una correcta programación. En particular, fomenta el desarrollo de pequeños módulos de código reubicable. *Código reubicable* es aquel que puede ejecutarse en *cualquier* posición de la memoria. Cada módulo puede diseñarse como si empezara en la posición cero. Cuando se ejecute, podrá cargarse con el inicio en cualquier posición que sea múltiplo de 16. Entonces, la dirección del segmento correspondiente se puede cargar en el segmento de código vía un *salto intersegmento* especial o llamada que comienza a ejecutar el código. Una vez este código se ha localizado, es muy fácil reubicarlo. De hecho el código máquina puede reubicarse dinámicamente trasladando el código (multiplicándolo por 16) y modificando el contenido del segmento código en consonancia.

Esto es muy útil en entornos de multiprogramación, donde diferentes trabajos o partes de trabajos tienen que cargarse y descargarse en diferentes posiciones de memoria ya que otros trabajos están ahora donde estaban ellos. La mayoría de los

nuevos sistemas operativos requieren esta habilidad para reubicar de forma rápida secciones de código de tamaño pequeño y medio.

JUEGO DE INSTRUCCIONES

Empezaremos esta sección explicando las instrucciones del 8086/8088 por grupos y la finalizaremos comparando estas instrucciones con las de las anteriores máquinas de 8 bits y las de los otros procesadores de 16 bits. Veremos el aumento de potencia de los procesadores de 16 bits respecto a sus predecesores de 8 bits, no sólo en el tamaño de los datos que pueden gestionar, sino también en sus otras características, como las nuevas operaciones de multiplicar y dividir, y las nuevas instrucciones para gestión de entornos de multi-proceso y de proceso en paralelo.

En general, las instrucciones del procesador de 16 bits se pueden clasificar en los siguientes grupos: 1) transferencia de datos, 2) aritmética entera binaria, 3) operaciones lógicas, 4) desplazamientos y rotaciones, 5) gestión de bits, 6) aritmética codificada en binario, 7) gestión de cadenas, 8) control del programa y 9) control del sistema. Estas categorías son algo arbitrarias, pero pueden aplicarse de forma general a los tres tipos de procesadores de 16 bits actuales: el Intel 8086/8088, el Zilog Z8000, y el Motorola MC68000.

Para las instrucciones del 8086/8088 usaremos los nemotécnicos de Microsoft dado que son probablemente los más populares. Con estos nemotécnicos muchas instrucciones indican (como parte de sus códigos de operación) si los operandos son en octetos o palabras o si la fuente son datos inmediatos. Una *B* extra en el nemotécnico de la operación indica modo octeto (datos de 8 bits), mientras que su ausencia indica modo de palabra (datos de 16 bits). Una *I* extra en el nemotécnico de la operación indica datos inmediatos en la fuente. Cuando la fuente es un dato inmediato de 8 bits, entonces aparecerán la *B* y la *I* (en este orden) en el nemotécnico de la operación. El uso de la *B* y de la *I* como parte del *código de operación* no debe confundirse con los modos de direccionamiento (descritos en la sección previa) y la forma en que éstos afectan al operando.

En esta sección, no desarrollaremos explícitamente el código máquina para cada una de las instrucciones. Esto lo haremos en profundidad en la sección siguiente; el apéndice D muestra una tabla con el código máquina para las diversas variantes de cada instrucción. El *MCS-86 Assembly Language Reference Guide* es muy adecuado para estos detalles.

Obsérvese la sintaxis de los operandos. Algunas instrucciones no tienen operandos, otras tienen uno y otras tienen dos. Para estas últimas, siempre se escribe primero el destino, separándolo con una coma del fuente. Como su nombre indica, el destino es donde se almacena el resultado, mientras que el fuente es la entrada y no es modificado por la operación.

Instrucciones de transferencia de datos

Las instrucciones de transferencia de datos son las encargadas de mover datos de un sitio a otro de la computadora como

pueden ser la memoria, el espacio de E/S y los registros de la CPU. Las instrucciones más comunes en los programas escritos en lenguaje ensamblador son típicamente éstas de transferencia de datos. Las instrucciones de transferencia son las siguientes:

MOV	destino, fuente	Transfiere una palabra de la fuente al punto de destino
MOVB	destino, fuente	Transfiere un octeto de la fuente al punto de destino
MOVI	destino, dato	Transfiere el dato (inmediato) de la fuente al punto de destino
MOVBI	destino, dato	Transfiere el dato (inmediato) al octeto de destino
XCHG	destino, fuente	Intercambia los contenidos de las palabras fuente y destino
XCHGB	destino, fuente	Intercambia los contenidos de los octetos fuente y destino
PUSH	fuente	Introduce la fuente en la pila
POP	destino	Extrae un elemento de la fuente y lo lleva al punto de destino
IN	fuente	Lleva a AX el contenido de la fuente (palabra)
INB	fuente	Lleva a AL el contenido de la fuente (octeto)
IN		Lleva a AX la posición DX (palabra)
INB		Lleva a AL la posición DX (octeto)
OUT	destino	Lleva a AX al punto de destino (palabra)
OUTB	destino	Lleva a AL al punto de destino (octeto)
OUT		Lleva a AX a la posición DX (palabra)
OUTB		Lleva a AL a la posición DX (octeto)
XLAT		Traduce (utilizando una tabla)
LEA	registro, fuente	Carga la dirección efectiva
LDS	registro, fuente	Carga DS y registro
LES	registro, fuente	Carga ES y registro

La instrucción MOV permite transferir datos de 8 bits (con el nemotécnico MOVB) o de 16 bits (MOV): 1) de registro a registro, 2) de registro a memoria, 3) de memoria a registro, 4) un dato inmediato a registro y 5) un dato inmediato a memoria. En

los casos 2), 3) y 5) la referencia a memoria puede hacerse en cualquiera de los 24 modos de direccionamiento de la CPU. Los casos 1), 2) y 3) utilizan el nemotécnico MOV (para 16 bits) o MOVB (datos de 8 bits); y los casos 4) y 5) utilizan el nemotécnico MOVI (datos de 16 bits) o MOVBI (datos de 8 bits).

Ejemplos de la instrucción MOV:

```
MOV    BX,CAT   ; equivalente a la
                ; LHLD CAT del 8080
MOVI   BX,CAT   ; equivalente a la
                ; LXI H,CAT del 8080
MOVB   AL,CAT   ; equivalente a la
                ; LDA CAT del 8080
MOVBI  AL,CAT   ; equivalente a la
                ; MVI A,3 del 8080
MOV    BP,SP    ; el puntero de base
                ; apunta a la pila
```

Es importante resaltar que en ciertos casos especiales de la instrucción MOV, el ensamblador produce (y el 8086/8088 acepta) códigos máquina distintos de lo que indica la situación general. Estos casos particulares se diseñaron para optimizar el rendimiento del 8086/8088 en situaciones frecuentes, aumentando la velocidad y disminuyendo el número de octetos del código máquina estándar. Es una situación similar a las contracciones del lenguaje hablado, como puede ser el «del» en vez de «de el», o «al» por «a el». Estas formas especiales del lenguaje máquina del 8086/8088 fueron diseñadas por los ingenieros del Intel.

Las formas optimizadas de la instrucción MOV del 8086/8088 son todas las de transferencia de datos entre el acumulador (AX para 16 bits o AL para 8) y memoria, con un solo modo de direccionamiento para la referencia a memoria. Con este tipo de direccionamiento no se tiene ni una base ni un índice; sólo un *decalage* que aparece como parte de la instrucción de una manera similar a los desplazamientos en los otros modos de direccionamiento. La dirección real se obtiene sumando el contenido del registro de segmentación (multiplicado por 16) al decalage, como es lo usual.

Hay otras formas de la instrucción MOV que transfieren datos a y desde los registros de segmentación.

Todas estas posibilidades distintas crean cantidades de códigos máquina; sin embargo, el lenguaje ensamblador es muy sencillo. El ensamblador analiza la selección de fuente y destino y escoge, automáticamente, el código máquina óptimo. Para hacer esto debe consultar rápidamente una serie de tablas.

La instrucción XCHG se utiliza para intercambiar los contenidos de dos registros, o una posición de memoria con un registro. Hay instrucciones XCHG tanto para 8 bits (XCHGB) como para 16 (XCHG). Un ejemplo de esta instrucción puede ser:

```
XCHG   BS,DS    ; es equivalente
                ; a la instrucción
                ; XCHG del 8080
```

Una forma especial optimizada genera códigos máquina de 1 octeto para intercambios entre el acumulador y los registros, aunque restringida a registros de 16 bits.

Las instrucciones PUSH y POP se usan para introducir y extraer datos de la pila del sistema. Sólo son válidas para datos de 16 bits. Algunos ejemplos son:

```
PUSH    AX      ; introduce en la pila
                ; el contenido de AX
POP     3       ; extrae de la pila
                ; el elemento superior
                ; y lo coloca en la posición
                ; 3 de memoria
```

Aunque en principio permiten cualquier tipo de direccionamiento existen formas optimizadas en el caso de que el operando sea un registro. Hay también formas especiales para el PUSH y POP de registros de segmentación. El capítulo del NDP 8087 incluye una discursión del funcionamiento de la pila, aunque la pila del NDP 8087 es interna, mientras que la de la CPU reside en memoria y utiliza un puntero de pila y un registro de segmentación de pila para formar el puntero al primer elemento de pila. Por ahora, basta con considerar que una pila es una especie de caja en la cual el último elemento introducido es el primero en salir, y que PUSH introduce un nuevo dato en la pila, mientras que POP lo extrae.

Las instrucciones IN y OUT transfieren datos a y desde la CPU y controladores de dispositivos de E/S (y celdas de memoria) cuyas direcciones están en el espacio de E/S del sistema. En realidad estas instrucciones tienen dos formas: una estática, en la cual la dirección del port se da como parte de la instrucción; y una dinámica, en la cual la dirección del port se encuentra en el registro DX y puede modificarse por programa. La versión dinámica permite direcciones de ports de 16 bits, pero la versión de 8 bits sólo permite direcciones de 8 bits para el port (como en el caso de los procesadores 8080 y 8085). El acumulador de la CPU (AX para datos de 16 bits y AL para datos de 8) sirve de destino en la instrucción IN, y de fuente en la instrucción OUT. Como ejemplo podemos ver:

```
INB     10H     ; transfiere 1 octeto
                ; del port 10H a AL
OUTB    11H     ; transfiere 1 octeto de AL
                ; al port 11H

MOVI    DX,10H  ; activa el port 10H
MOV     AX,BX   ; deja los datos preparados
OUT             ; transfiere los datos
                ; al port de E/S
```

La instrucción XLAT (traduce) realiza la traducción a través de una tabla de búsqueda. La tabla debe cargarse en memoria y la dirección de comienzo (base) se guarda en el registro BX antes de utilizar esta instrucción. Si los octetos de entrada de la tabla son a(0), a(1), a(2)..., a(255), un valor *i* en AL antes de la instrucción se traduce (se reemplaza por) el valor *a(i)* al ejecutarse XLAT.

Los mecanismos de funcionamiento de XLAT son los siguientes: los contenidos del registro BX y de AL se suman para obtener la dirección de una posición de memoria, y el contenido de dicha posición se copia en AL. La instrucción XLAT traduce datos sólo de 8 bits (en AL), por tanto restringidos a un rango de 0 a 255. Aquí se presenta un ejemplo de XLAT:

```
MOVI    BX,TABLE   ; apunta a la tabla "TABLE"
MOVBI   AL,'*'     ; pone un '*' (en ASCII)
XLAT               ; AL tiene su nuevo código
```

La instrucción LEA (carga de la dirección efectiva) se utiliza para cargar el desplazamiento de cualquier operando especificado en uno de los 24 modos de direccionamiento. LEA coloca este desplazamiento en el registro de destino. Los procesadores de 16 bits Zilog y Motorola tienen ambos instrucciones equivalentes a ésta, mientras que los de 8 bits como el 8080/8085, Z80 y 6502 no las tienen. Como ejemplo:

```
LEA   BX,COLOR[DI]   ; BX especifica la
                     ; dirección de COLOR
                     ; + (DI)
```

La instrucción LDS (carga el segmento de datos) se utiliza para cargar el segmento de datos de cualquier registro de 16 bits (no de segmentación) con los datos de memoria. Los contenidos de la fuente se colocan en el registro seleccionado (registro destino), y el contenido del registro siguiente se introduce en el registro de segmentación de datos. La instrucción LES (carga segmento extra) funciona de forma parecida, excepto que en vez del registro de segmentación de datos se utiliza un registro de segmentación extra. Así por ejemplo:

```
LDS   SI,SPOINTER   ; obtener el puntero
                    ; fuente de memoria
LES   DI,DPOINTER   ; obtener el puntero
                    ; destino de memoria
```

Aritmética binaria entera

Las operaciones en aritmética binaria entera permiten a la CPU realizar cálculos con números enteros positivos y negativos, estos últimos en complemento a dos. Dichas instrucciones incluyen:

NEG	destino	Hace negativo el número que encuentra en la posición de destino
NEGB	destino	Hace negativo el octeto de destino
ADD	destino, fuente	Suma fuente y destino (palabras)
ADDB	destino, fuente	Suma fuente y destino (octetos)
ADDI	destino, dato	Suma el dato inmediato a destino (palabra)
ADDBI	destino, dato	Suma el dato inmediato a destino (octeto)
ADC	destino, fuente	Suma la fuente + acarreo a destino (palabra)
ADCB	destino, fuente	Suma la fuente + acarreo a destino (octeto)

ADCI	destino, dato	Suma dato + acarreo a destino (palabra)
ADCBI	destino, fuente	Suma dato + acarreo a destino (octeto)
SUB	destino, fuente	Resta la fuente de destino (palabra)
SUBB	destino, fuente	Resta la fuente de destino (octeto)
SUBI	destino, dato	Resta dato de destino (palabra)
SUBBI	destino, dato	Resta dato de destino (octeto)
SBB	destino, fuente	Resta fuente + acarreo de destino
SBBB	destino, fuente	Resta fuente + acarreo de destino (octeto)
SBBI	destino, dato	Resta dato+acarreo de destino 8 bits
SBBBI	destino, dato	Resta dato + acarreo de destino (octeto)
MUL	fuente	Multiplicación de números positivos de 16 bits
MULB	fuente	Multiplicación de números positivos de 8 bits
IMUL	fuente	Multiplicación de números con signo de 16 bits
IMULB	fuente	Multiplicación de números con signo de 8 bits
DIV	fuente	División de números positivos de 16 bits
DIVB	fuente	División de números positivos de 8 bits
IDIV	fuente	División de números con signo de 16 bits
IDIVB	fuente	División de números con signo de 8 bits
CBW	fuente	Pasa de octeto a palabra
CWD	fuente	Pasa de octeto a doble-palabra
INC	destino	Incrementa destino (palabra)
INCB	destino	Incrementa destino (octeto)
DEC	destino	Decrementa destino (palabra)
DECB	destino	Decrementa destino (octeto)

La instrucción NEG convierte en negativo (rotación de complemento a dos) el valor de destino. Existen versiones para 8 bits (NEGB) y para 16 (NEG). Algunos ejemplos:

```
NEG     AX      ; complemento a dos de AX
NEGB    AL      ; complemento a dos de AL
```

Las instrucciones ADD y SUB realizan las operaciones de suma y resta respectivamente de números, o bien *con signo* (complemento a dos) o *sin signo* (positivos). En ADD, la fuente se

suma a destino; y con SUB, la fuente siempre se resta de destino. Como ejemplos:

```
ADDB    AL,CL   ; equivalente a ADD C
                ; en el 8080
ADD     BX,DX   ; equivalente a DAD D
                ; en el 8080
ADBBI   AL,5    ; equivalente a ADI 5
                ; en el 8080
ADCBI   AL,7    ; equivalente a ACI 7
                ; en el 8080
```

Todas las operaciones aritméticas del 8086/8088 trabajan con *seis* tipos de datos diferentes, con enteros positivos o con signo de 8 ó 16 bits y con dos tipos de código decimal codificado en binario (que veremos más tarde). Tanto la multiplicación como la división, suma y resta, soportan los seis tipos de aritmética del 8086/8088 (incluyendo la decimal codificada binaria).

Debido a la forma en que funciona la aritmética de complemento a dos, las operaciones reales para la suma y la resta son iguales tanto para números positivos como números con signo. La única diferencia es el punto donde ocurre la capacidad excedida, y esto puede controlarse por programa mirando el estado de los indicadores. Con aritmética de números sin signo (positivos), la capacidad excedida se da siempre que hay un *acarreo* de salida del bit más significativo (de la izquierda), de modo que el *indicador de acarreo* (CF) se pone a 1 siempre que ocurre un exceso de capacidad en números *sin signo*. Por ejemplo, obsérvense las siguientes sumas binarias:

```
  0101              1011
 +0110              0111
 ------            ------
  1011              0010
no hay acarreo     acarreo
```

En el primer caso, hemos sumado 5 y 6 y nos ha dado 11. El resultado es bueno ya que no ha habido acarreo de salida en el dígito más significativo. El indicador de acarreo estará a cero. En el segundo caso, la suma de 7 y 11 ha dado 2 (todo esto en binario, por supuesto). El acarreo de salida nos indica que el resultado es erróneo (el bit de acarreo de salida se emplea para poner a 1 el indicador CF). El indicador de acarreo avisa en ambos casos si la operación es o no errónea.

En números con signo (positivos y negativos), la situación es algo más compleja. En este caso será el *indicador de desbordamiento* (OF) el que nos detecte la operación errónea. Básicamente, en aritmética de números con signo, ocurre un error siempre que el bit de signo del resultado no sea correcto. Por ejemplo, supongamos que queremos sumar dos números positivos. Si el resultado de la suma es demasiado grande, ¡el signo del resultado aparecerá negativo! La CPU posee circuitos específicos que detectan estas situaciones y ponen a 1 el indicador de capacidad excedida. Observemos las siguientes sumas de números de 4 bits.

0101	1011
+0110	0111
------	------
1011	0010
desbordamiento	no desbordamiento

En el primer caso se suma +5 con +6 y se obtiene un −5. Evidentemente, la operación ha sido errónea, cosa que se detecta observando que los signos de los sumandos eran positivos y el del resultado es negativo. En el segundo caso, hemos sumado −5 y 7, y obtenido el resultado correcto 2. Cuando los dos operandos tienen signos distintos, nunca se excede la capacidad. No puede darse ningún error en este caso.

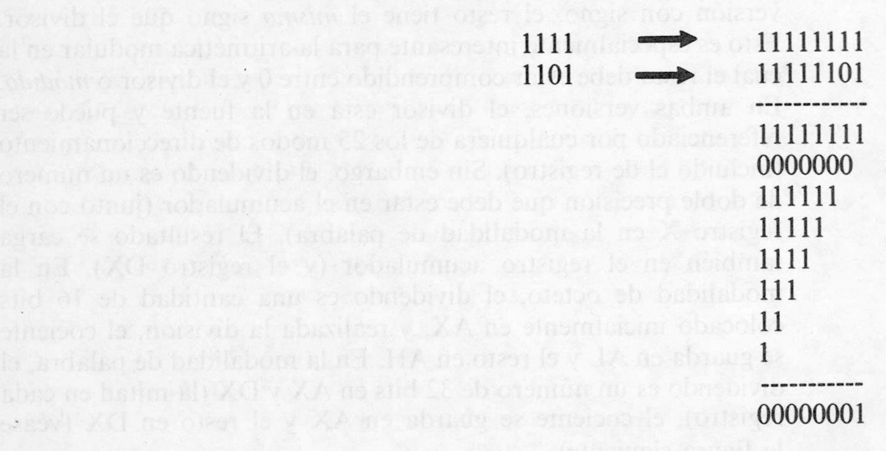
Las instrucciones ADC y SBB trabajan igual que las ADD y SUB, salvo que el contenido del indicador de acarreo interviene (entra) también en el cálculo. Estas dos instrucciones se usan en rutinas software de operaciones multidígito, por ejemplo, cuando queremos realizar operaciones de 16 bits en máquinas de 8.

Las instrucciones ADD, SUB, ADC y SBB permiten operandos de 8 ó 16 bits con: 1) registro fuente y registro destino, 2) registro fuente y destino en memoria, 3) fuente en memoria y registro destino, 4) dato inmediato para la fuente y registro destino y 5) dato inmediato para la fuente y destino en memoria. La referencia a memoria en los casos 2), 3) y 5) puede hacerse con cualquiera de los 24 modos de direccionamiento. En contraste con la instrucción MOV, los registros de segmentación nunca pueden aparecer como operandos; pero, al igual que MOV, existen optimizaciones especiales del código máquina en ciertos casos. El caso de operación entre un dato inmediato y el registro acumulador (un subcaso de 4), existe una versión óptima que ocupa 1 octeto menos de código máquina que lo que ocuparía en el caso general de operación de un dato inmediato con *cualquier* registro. Hay también un modo especial de 4) y 5) en el cual un dato inmediato de 8 bits se convierte automáticamente en 16 bits antes de la operación.

Existen dos versiones de la operación de multiplicación: MUL para los números con signo, o IMUL para los positivos. Para ver la diferencia, consideremos el siguiente ejemplo: Supongamos que queremos multiplicar los números 1111 por 1101. El algoritmo usual de multiplicación funcionaría como sigue:

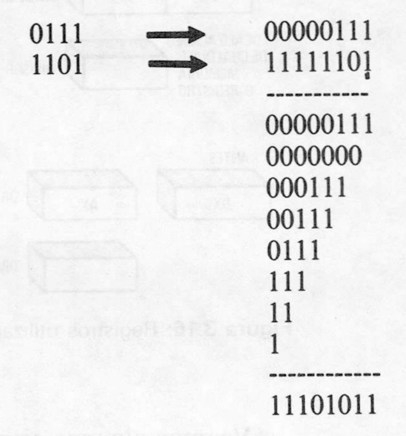
```
   1111
   1101
   ------
   1111
  0000
 1111
1111
-----------
1000011
```

Con números positivos, el número binario 1111 representa el valor decimal 15, y el 1101 representa el 13. El resultado, 11000011, es el número decimal 195. El resultado es correcto. Sin embargo, si los números tienen signo y se sigue la representación de complemento a dos, el número 1111 representa el valor −1; el número 1101 representa −3, y el resultado, 11000011, representa el valor −61; resultado claramente erróneo. Hay varias maneras de corregir el error. Una forma es aumentar el número de bits de los dos factores a 8 antes de realizar el producto, y utilizar la aritmética de 8 bits a continuación:



```
1111  ──►  11111111
1101  ──►  11111101
           ----------
           11111111
           0000000
           111111
           11111
           1111
           111
           11
           1
           ----------
           00000001
```

En este caso el resultado es 00000011, que representa el valor decimal +3 previsto. Téngase en cuenta que ambos factores eran negativos, y se expandieron colocando 1 en los bits de valor más alto. Los números positivos (no negativos) se extienden colocando 0 en las posiciones más significativas; por ejemplo,



```
0111  ──►  00000111
1101  ──►  11111101
           ----------
           00000111
           0000000
           000111
           00111
           0111
           111
           11
           1
           ----------
           11101011
```

Esta vez se obtiene un −21, igual al valor esperado de multiplicar +7 por −3.

Tanto para números con signo como sin signo, la fuente se multiplica por el destino, y el resultado se guarda en el acumulador (extendido, si es necesario, al registro DX). Más concretamente, hay modalidades de palabra y de octeto para las versiones

con y sin signo. En el modo de octeto (8 bits), el resultado (16 bits) se guarda en el registro acumulador AX de 16 bits; y en el modo de palabra (16 bits), el resultado de 32 bits se guarda la mitad en el registro AX y la otra mitad en el registro DX. El resultado tiene el doble de *precisión* (número de dígitos) que los operandos, y en este sentido se dice que el resultado es de *doble precisión*.

La instrucción DIV posee también versiones para números con o sin signo, y modalidades de palabra u octeto. La instrucción DIV devuelve dos resultados, un cociente y el resto. En la versión con signo, el resto tiene el *mismo* signo que el divisor. Esto es especialmente interesante para la aritmética modular en la cual el resto debe estar comprendido entre 0 y el divisor o *módulo*. En ambas versiones, el divisor está en la fuente y puede ser referenciado por cualquiera de los 25 modos de direccionamiento (incluido el de registro). Sin embargo, el dividendo es un número de doble precisión que debe estar en el acumulador (junto con el registro X en la modalidad de palabra). El resultado se carga también en el registro acumulador (y el registro DX). En la modalidad de octeto, el dividendo es una cantidad de 16 bits colocado inicialmente en AX, y realizada la división, el cociente se guarda en AL y el resto en AH. En la modalidad de palabra, el dividendo es un número de 32 bits en AX y DX (la mitad en cada registro), el cociente se guarda en AX y el resto en DX (véase la figura siguiente).

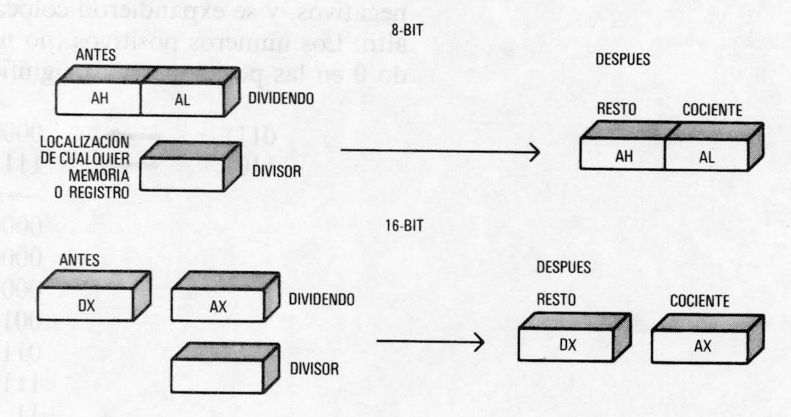


Figura 3.16: Registros utilizados en la división.

Veamos algunos ejemplos de multiplicaciones y divisiones:

```
MOVBI   AL,3    ; comenzar con 3
MOVBI   CL,5    ; y 5
MULB    CL      ; el resultado 15 aparece en AX
MOVBI   CL,6    ; el divisor es 6
DIVB    CL      ; el cociente 2 aparece en AL
                ; y el resto 3 en AH
```

Las instrucciones de conversión de octeto a palabra y de palabra a doble palabra (CBW y CWD) son útiles a la hora de expandir tamaños de palabra. Según uno de los diseñadores del 8086, Stephen Morse, a estas instrucciones originalmente se les dio el nombre de SEX, por EXtensión del Signo (Sign EXtend). El nombre se cambió por el más conservador de CBW y CWD antes de salir al mercado. El propósito fundamental de estas instrucciones es el de preparar los números para la división. Por ejemplo, si queremos dividir un número de 16 bits, NUMBER, por otro número de 16 bits, DIVISOR, el DIVISOR puede colocarse en casi cualquier sitio (salvo tal vez en AX, DX, IP y los registros de segmentación), pero el dividendo NUMBER debe colocarse obligatoriamente en AX. A continuación, NUMBER debe extenderse a 32 bits, la mitad inferior en AX y la mitad superior en DX. Si NUMBER es positivo, la extensión consiste simplemente en colocar 0 en DX; pero si NUMBER es negativo, todos los bits de DX deben ponerse a 1. Esto es precisamente lo que hace de una manera automática la instrucción CWD. La CBW realiza la misma extensión del registro AL de 8 bits en AX, de 16. Aquí podemos ver un ejemplo:

```
MOVB    AL,XCRD   ; obtener la coordenada X
CBM               ; y extenderla a 16 bits
MOVBI   CL,FAC    ; obtener el factor de escala
IDIV    CL        ; dividir con signo por
                  ; el factor de escala
MOVB    XCRD,AL   ; devolver XCRD
```

Las instrucciones INC y DEC se usan para incrementar (sumar 1) o decrementar (restar 1) el valor de destino. Se puede utilizar cualquiera de los 25 modos de direccionamiento posibles para especificar el punto de destino. INC y, especialmente, DEC son útiles a la hora de contar el número de veces que se ejecuta un bucle, aunque el 8086/8088 tiene una instrucción específica (LOOP) y más potente para ello. Veamos algunos ejemplos:

```
INCB   AL  ; equivalente a INR A
           ; del 8Ø8Ø

INC    BX  ; no es equivalente a
           ; INX H del 8Ø8Ø
           ; porque no se preservan
           ; los indicadores
```

Operaciones lógicas

Las operaciones lógicas se utilizan para poner a 1, borrar (poner a 0) y cambiar o examinar bits individuales de la computadora. La ventaja principal de trabajar en lenguaje ensamblador es que se consigue un mejor control de la computadora, y estas operaciones pueden ser una gran ayuda en este aspecto. Posibles aplicaciones pueden ser el examinar bits de estado para la E/S y para el control del sistema, o poner a 1 ciertos bits de control. Otro uso específico de estas instrucciones puede ser el desarrollo de un ensamblador. En este caso, ¡los bits de control

controlan la CPU! En la siguiente sección veremos los bits individuales del lenguaje máquina del 8086/8088. Pasemos antes a ver cuáles son las operaciones lógicas.

NOT	destino	Niega (hace la función NOT) el destino (palabra)
NOTB	destino	Niega (hace la función NOT) el destino (octeto)
AND	destino, fuente	Producto lógico (AND) de destino y fuente (palabra)
ANDB	destino, fuente	Producto lógico de destino y fuente (octeto)
ANDI	destino, dato	Producto lógico de destino y el dato inmediato (palabra)
ANDBI	destino, dato	Producto lógico de destino y dato inmediato (octeto)
OR	destino, fuente	Suma lógica (OR) de destino y fuente (palabra)
ORB	destino, fuente	Suma lógica (OR) de destino y fuente (octeto)
ORI	destino, dato	Suma lógica (OR) de destino y dato inmediato (palabra)
ORBI	destino, dato	Suma lógica (OR) de destino y dato inmediato (octeto)
XOR	destino, fuente	XOR de destino y fuente (palabra)
XORB	destino, fuente	XOR de destino y fuente (octeto)
XORI	destino, dato	XOR de destino y dato inmediato (palabra)
XORBI	destino, dato	XOR de destino y dato inmediato (octeto)

Las operaciones lógicas del 8086/8088 (y de todos los procesadores conocidos) trabajan independientemente bit a bit según las siguientes tablas de verdad:

X	NOT X		X	Y	X AND Y	X OR Y	XX OR Y
0	1		0	0	0	0	0
1	0		0	1	0	1	1
			1	0	0	1	1
			1	1	1	1	0

Tabla 3.2: Tablas de verdad de las operaciones lógicas.

Aunque las instrucciones lógicas operan bit a bit, en el 8086/8088, los operandos son octetos o palabras completas. Es

decir, si se quiere cambiar un bit particular, es necesario primero localizarlo dentro de algún octeto o palabra de memoria, o en algún registro de la CPU. Muchas veces es interesante el que la operación lógica modifique todos los bits del octeto o palabra. Por ejemplo, si queremos complementar (negar) todos los bits de una cantidad de 8 bits, 10110111, guardada en el acumulador AH, basta con hacer:

```
NOTB  AH   ; complementa los 8 bits de AH
```

Tras la ejecución de esta instrucción, AH contendrá el valor 01001000. Otro ejemplo es el producto lógico (AND) de dos cantidades de 8 bis, 11001100 y 10101010. El producto lógico se obtiene haciendo el AND de cada bit de la primera cantidad y el correspondiente bit de la segunda, dando en este caso 10001000. Esto es,

```
      11001100
AND   10101010
      --------
      10001000
```

En cualquier caso, el resultado de la operación lógica se guarda en destino. Con combinaciones de operación lógicas se puede cambiar realmente el valor de cualquier bit particular del octeto o palabra. Por ejemplo, si se quiere poner a cero el segundo bit más significativo de la palabra almacenada en AX, se puede hacer:

```
ANDI  AX,1011111111111111   ; deja igual
                            ; todos los bits
                            ; salvo el segundo
```

Igual se podía haber escrito:

```
ANDI  AX,0BFFFFH   ; todo unos salvo
                   ; el segundo
                   ; de la izquierda
```

La instrucción NOT tiene un solo operando, el destino, que puede referenciarse por cualquiera de los 25 modos de direccionamiento. No existe ninguna optimización de código máquina para esta instrucción. Las instrucciones AND, OR y XOR poseen dos operandos, con la misma estructura de direccionamiento que las instrucciones aritméticas ADD, SUB, ADC y SBB estudiadas anteriormente. La única diferencia es que, como éstas son operaciones lógicas, no existe la modalidad de extensión de 8 a 16 bits.

Desplazamientos y rotaciones

Las operaciones de *desplazamiento* y *rotación* permiten mover bits a la izquierda o a la derecha en celdas de memoria (en memoria central o en registros de la CPU) de la computadora. No

hay definiciones aceptadas universalmente para nombrar la gran cantidad de variaciones posibles de estas operaciones, pero los nemotécnicos del 8086/8088 constituyen una buena nomenclatura de estas instrucciones.

La figura siguiente muestra las versiones de 8 bits de todas ellas:

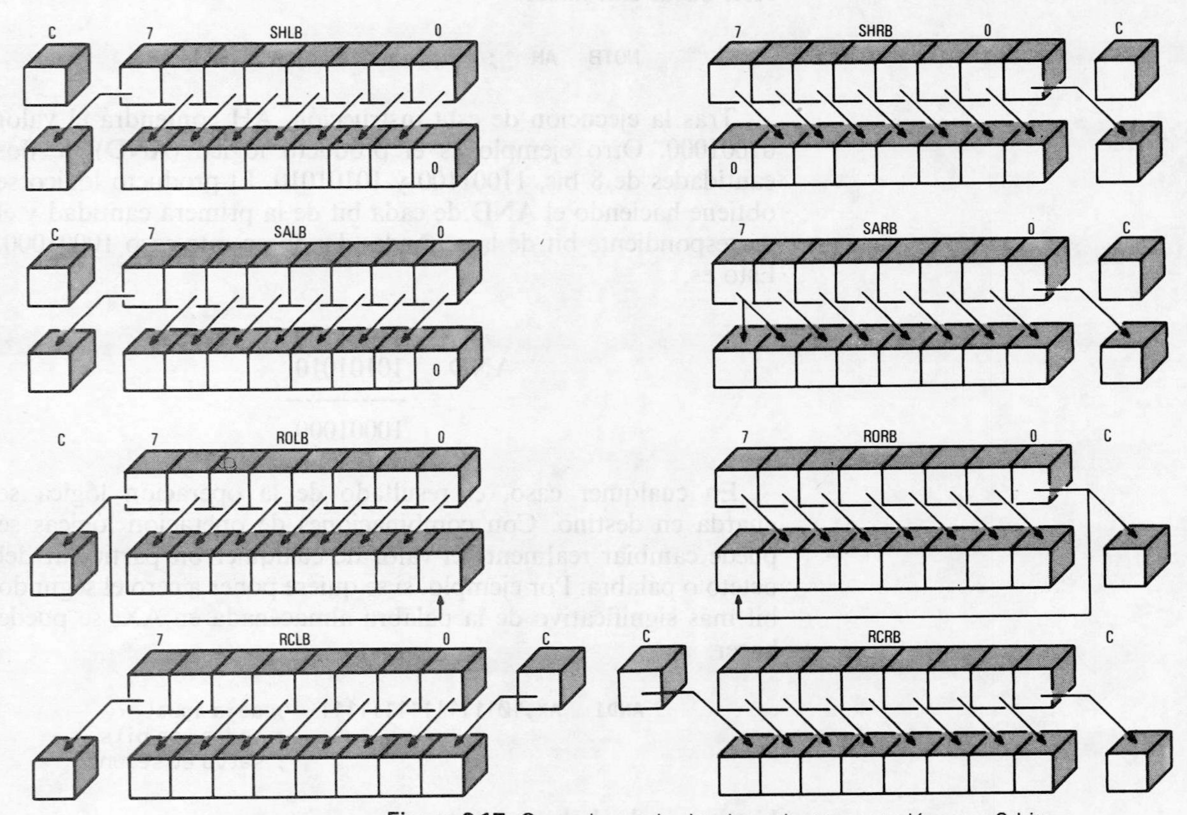


Figura 3.17: Operaciones de desplazamiento y rotación para 8 bits.

Básicamente, una rotación mueve (a izquierda o derecha) los bits de la palabra u octeto de una forma *circular,* como si la palabra u octeto se hubiera curvado hasta conectar el bit de más a la derecha con el de más a la izquierda; mientras que un desplazamiento mueve los bits de una forma *lineal*. En cualquiera de los casos, Intel ofrece dos variantes.

Para la rotación, hay una rotación pura, y otra que incluye el bit de acarreo. Para los desplazamientos, existe el desplazamiento *lógico* y el desplazamiento *aritmético*.

En el desplazamiento lógico, los bits que entran a la celda de memoria son 0.

En el aritmético, un desplazamiento a la derecha divide por dos el contenido de la celda, y un desplazamiento a la izquierda lo multiplica por dos. Es decir, los desplazamientos aritméticos a derecha e izquierda realizan la división o producto respectivamente del valor de la celda de memoria por la base del sistema de

numeración empleado, en este caso el 2. Fijémonos que multiplicar o dividir por 10 en el sistema de numeración decimal equivale a desplazar los dígitos a izquierda o derecha. Hecha esta distinción entre desplazamientos lógicos y aritméticos, permítasenos decir que en realidad un desplazamiento lógico a la izquierda es exactamente lo mismo que un desplazamiento aritmético a la izquierda. Sin embargo, con los desplazamientos a la derecha no ocurre lo mismo. El *desplazamiento aritmético a la derecha* pone 0 en el bit más significativo (el de signo) si el número era positivo, y 1 si el número era negativo. Dicho de otra manera, el desplazamiento aritmético a la derecha preserva el signo. Después de todo, es lógico desde el momento que al dividir el número por 2 se conserva el valor del signo. Es importante resaltar que el desplazamiento aritmético a la derecha *redondea por defecto* el resultado. Así, el desplazamiento de 3 da 1, que es $3/2=1{,}5$ redondeado por defecto, y el desplazamiento de -3 da -2, que es $-3/2=1{,}5$ redondeado por defecto. Cosa curiosa, si se hace un desplazamiento aritmético a la derecha da -1, ¡vuelve a dar -1!

Veamos estas instrucciones:

SHL	destino	Desplazamiento lógico a la izquierda (una posición)
SHL	destino, CL	Desplazamiento lógico a la izquierda (CL posiciones)
SHLB	destino	Desplazamiento lógico a la izquierda de un octeto (una posición)
SHLB	destino, CL	Desplazamiento lógico a la izquierda de un octeto (CL posiciones)
SHR	destino	Desplazamiento lógico a la derecha (una posición)
SHR	destino, CL	Desplazamiento lógico a la derecha (CL posiciones)
SHRB	destino	Desplazamiento lógico a la derecha de un octeto (una posición)
SHRB	destino, CL	Desplazamiento lógico a la derecha de un octeto (CL posiciones)
SAL	destino	Desplazamiento aritmético a la izquierda (una posición)
SAL	destino, CL	Desplazamiento aritmético a la izquierda (CL posiciones)
SALB	destino	Desplazamiento aritmético a la izquierda de un octeto (una posición)
SALB	destino, CL	Desplazamiento aritmético a la izquierda de un octeto (CL posiciones)

SAR	destino	Desplazamiento aritmético a la derecha (una posición)
SAR	destino, CL	Desplazamiento aritmético a la derecha (CL posiciones)
SARB	destino	Desplazamiento aritmético a la derecha de un octeto (una posición)
SARB	destino, CL	Desplazamiento aritmético a la derecha de un octeto (CL posiciones)
ROL	destino	Rotación a la izquierda (una posición)
ROL	destino, CL	Rotación a la izquierda (CL posiciones)
ROLB	destino	Rotación de un octeto a la izquierda (una posición)
ROLB	destino, CL	Rotación de un octeto a la izquierda (CL posiciones)
ROR	destino	Rotación a la derecha (una posición)
ROR	destino, CL	Rotación a la derecha (CL posiciones)
RORB	destino	Rotación de un octeto a la derecha (una posición)
RORB	destino, CL	Rotación de un octeto a la derecha (CL posiciones)
RCL	destino	Rotación a la izquierda con acarreo incluido (una posición)
RCL	destino, CL	Rotación a la izquierda con acarreo incluido (CL posiciones)
RCLB	destino	Rotación a la izquierda de un octeto con acarreo incluido (una posición)
RCLB	destino, CL	Rotación a la izquierda de un octeto con acarreo incluido (CL posiciones)
RCR	destino	Rotación a la derecha con acarreo incluido (una posición)
RCR	destino, CL	Rotación a la derecha con acarreo incluido (CL posiciones)
RCRB	destino	Rotación a la derecha de un octeto con acarreo incluido (una posición)
RCRB	destino, CL	Rotación a la derecha de un octeto con acarreo incluido (CL posiciones)

Como puede verse, el 8086/8088 posee una gama completa de instrucciones de desplazamiento y rotaciones.

Cada instrucción de rotación o desplazamiento tiene dos versiones: una *estática*, de un solo paso, y una *dinámica*, de varios pasos. El número de pasos (número de veces que se realiza la operación) recibe el nombre de *contador de desplazamiento*. En la primera versión, el único operando es el destino, que contiene la cantidad a desplazar o rotar, y el contador de desplazamientos tiene implícitamente el valor 1.

En la versión dinámica de varios pasos, el contador de pasos está en el registro CL, y hay dos operandos: el primer operando es el destino (como en el caso anterior) y el segundo es explícitamente el registro CL.

Veamos algunos ejemplos:

```
ROR    DX     ; rotación a la derecha del
              ; registro DX (16 bits)
RCR    DX     ; rotación a la derecha de
              ; DX y CF (17 bits)
RORB   AL     ; equivalente a RRC
              ; del 8Ø8Ø
RCRB   AL     ; equivalente a RAR
              ; del 8Ø8Ø

MOVBI  CL,4   ; pon 4 en el contador
              ; de desplazamientos
ROR    AX,CL  ; rota a la derecha 4 posiciones
```

Los desplazamientos aritméticos merecen un comentario aparte. La instrucción SAL equivale a multiplicar por una potencia de 2. El exponente viene determinado por el contador de desplazamientos. Por ejemplo, si el contador es 3, el valor de destino se multiplica tres veces por 2; esto es, queda multiplicado un total por:

$$2 * 2 * 2 = 2^3 = 8$$

La instrucción SAR equivale a dividir por la potencia de 2 que indica el contador de desplazamientos. Si el contador es 1, el resultado de la operación es una división por 2. Siempre que se tenga que multiplicar o dividir por potencias de 2, es recomendable utilizar estas instrucciones porque resultan bastante más rápidas. A continuación damos algunos ejemplos:

```
SALB   AL     ; doblar el valor del acumulador
              ; (8 bits)
SARB   AL     ; dividir por 2 el acumulador
              ; (8 bits)
MOVBI  CL,3   ; poner 3 en el contador
              ; de desplazamiento
SAL    DX,CL  ; multiplicar DX por 8
```

Tratamiento de bits

No hay verdaderas operaciones de bits en el 8086/8088. Una *operación de bits* es aquella en la que se modifica (se pone a 1, se borra o se complementa) un bit explícitamente *nombrado*. Tales

operaciones se pueden simular con las instrucciones lógicas. Por ejemplo, si se quiere complementar el bit número 5 (¡recordemos que se empieza a contar por cero!) del registro AL, puede hacerse de la siguiente manera:

```
XORBI   AL,32   ; 32 es 2⁵
```

Nótese que la fuente es 00010000, que tiene un 1 en la quinta posición y ceros en las restantes. El resultado de la operación puede comprobarse con la tabla de verdad de la función XOR. En general, para conseguir aislar la *i*-ésima posición de un número, se utiliza la potencia *i*-ésima de 2. Es decir, para modificar el bit 0, usaremos $2^0=1$; para modificar el bit 1, usaremos $2^1=2$, etc...

Con *macros* (instrucciones en lenguaje ensamblador definidas por el usuario), se pueden escribir instrucciones particulares de tratamiento de bits. Por ejemplo, una macro para complementar cualquier bit de una palabra podría definirse así:

```
BITCOM  MACRO   X,I           ; define la macro
        XORI    X,(1 SHL I)   ; el ensamblador aisla
                              ; el bit I-ésimo
        ENDM                  ; ¡y eso es todo!
```

para usarla bastaría con hacer:

```
BITCOM  AX,5   ; equivalente a XORI AX,32
```

Cuando el ensamblador ve esta línea, sustituye el X por AX y la I por 5. La expresión (1 SHL I) se convierte en (1 SHL 5), que vale 32. En consecuencia, la instrucción BITCOM se interpreta como XORBI AL,32, como ya dijimos.

El uso de macros permite trabajar con bits de una forma parecida al Zilog Z8000. Motorola tiene un juego de instrucciones de tratamiento de bits bastante más elaborado y completo que Zilog. Motorola siempre verifica cualquier bit (activando o desactivando un indicador) antes de modificarlo.

Si se quisiera hacer por software (podría ser mediante una macro), el Z8000, o el 8086/8088 necesitaría el doble de tiempo que el Motorola.

Aritmética decimal codificada en binario

Las operaciones de aritmética decimal codificada en binario incluyen:

AAA	Ajuste ASCII para la suma
DAA	Ajuste decimal para la suma
AAS	Ajuste ASCII para la resta
DAS	Ajuste decimal para la resta
AAM	Ajuste ASCII para la multiplicación
AAD	Ajuste ASCII para la división

Estas operaciones se usan junto a las de suma, resta, multiplicación y división de números binarios, ADD, ADC, SUB, SBB,

MUL, IMUL, DIV e IDIV, para producir resultados en código decimal codificado en binario (BC) empaquetados o desempaquetados.

En la versión *BCD desempaquetada,* todo octeto de datos puede tomar sólo los valores de 0 a 9, y contiene un solo dígito de un número decimal. Con este sistema, por lo tanto, para representar un número decimal se necesitan tantos octetos como cifras tenga el número. En la versión *BCD empaquetada,* cada octeto se considera dividido en dos partes, de modo que cada parte contiene un dígito decimal de 0 a 9. Los valores de las cifras pueden leerse directamente a partir de la representación hexadecimal. Por ejemplo, si un octeto contiene el valor 34, el cuarteto superior contiene un 3 y el cuarteto inferior un 4. El octeto contiene la versión BCD empaquetada del número 34.

Las versiones desempaquetadas de la representación decimal codificada en binario usan las operaciones de ajuste ASCII para la suma (AAA), resta (AAS), multiplicación (AAM) y división (AAD). Las versiones empaquetadas emplean los ajustes decimales DAA (para la suma) y DAS (para la resta).

Veamos ahora cómo trabajan estas operaciones. Comencemos por la versión BCD desempaquetada. Supongamos que queremos sumar 7 más 5, ambos como números BCD desempaquetados. La suma binaria habitual sumaría un octeto con el valor 7 a un octeto con el valor 5, y devolvería el valor hexadecimal C, que no es ningún resultado BCD desempaquetado válido. La respuesta correcta debe ser 2 con un arrastre de 1. La instrucción AAA arregla las cosas de la siguiente manera: suponiendo que el resultado de la suma binaria se guarda en el registro AL, la instrucción AAA verifica si el cuarteto inferior de AL es demasiado grande (mayor que 9). Si lo es, suma 6 a AL, 1 a AH, y pone el cuarteto superior de AL a 0. En nuestro ejemplo, la instrucción AAA realiza la siguiente secuencia de operaciones (notación hexadecimal):

paso 0) AL contiene C que es demasiado grande
paso 1) suma 6 a AL. AL contendrá C+6=12
(cuidado, empleamos notación hexadecimal)
paso 2) incrementa AH
paso 3) borra el cuarteto superior de AL, dejando en AL el valor 2, que es el resultado correcto (dígito menos significativo).

No hemos dicho nada de cómo intervienen los indicadores AF y CF porque frecuentemente se puede ignorar en una aplicación real.

En el capítulo 7 hay un ejemplo de cómo puede utilizarse la instrucción AAA junto con la ADD y ADC para realizar cálculos multidígitos. En dicho ejemplo, el dígito siguiente se coloca en AM, de manera que puede recibir cualquier arrastre generado al ajustar el dígito en curso.

Veamos a continuación la versión empaquetada. Supongamos

que tenemos dos números de dígitos, el 34 y el 29, almacenados en dos octetos distintos en el formato BCD desempaquetado, y supongamos que queremos sumarlos. La instrucción ADD haría lo siguiente:

```
 34
+29
----
 5D
```

El resultado de la suma binaria normal es de nuevo incorrecto. El resultado correcto es 63. El cuarteto de la derecha es demasiado grande (excede en 16 el valor correcto), y el de la izquierda es demasiado pequeño (le falta 1). La instrucción DAA arregla las cosas sumando 6 al resultado obtenido. Más concretamente, la instrucción DAA ejecuta el siguiente microprograma sobre el valor del registro AL:

Si el cuarteto inferior de AL es mayor que 9,
    entonces suma 6 a AL,
fin del si.

Si el cuarteto superior de AL es mayor que 9,
    entonces suma 60 (hexadecimal) a AL,
        pone a 1 el indicador de arrastre,
fin del si.

De nuevo hemos ignorado los indicadores AF y CF (excepto al final). En nuestro ejemplo (34+29), se realizarían los siguientes pasos:

paso 0) AL contiene 5D (hexadecimal)
paso 1) suma 6 a AL, de forma que AL contiene 5D+6=63
    (en hexadecimal. En BCD es correcto)
paso 2) el cuarteto superior de AL es correcto, así que para.

Nótese que el hecho de sumar 6 automáticamente aumenta en 1 el valor del cuarteto superior.

En una aplicación real en la que quisiéramos sumar dos números multidígitos, comenzaríamos a trabajar de derecha a izquierda como es lo usual, pero no con dos dígitos cada vez. Cada dos dígitos (un octeto), utilizaríamos la instrucción ADC, y la DAA a continuación. Si la instrucción DAA anterior hubiese generado un arrastre, la instrucción ADC lo tendría en cuenta y lo sumaría correctamente al par de dígitos en curso. Como la instrucción DAA sólo trabaja con el registro AL, sería necesario ir metiendo y sacando los contenidos de AL conforme se fueran procesando los pares de dígitos.

Las instrucciones AAS y DAS realizan las mismas operaciones para la resta. La multiplicación y división sólo trabajan con versiones desempaquetadas de un dígito por octeto. La instruc-

ción AAM funciona de la siguiente manera: El contenido de AL se divide por 10; el cociente se guarda en AH y el resto en AL. Aunque parezca mentira, la instrucción AAM es realmente útil en productos multidígitos desempaquetados. No entraremos aquí en detalles, pero sí haremos un par de comentarios que creemos interesantes acerca de esta instrucción. Primero, para números entre 0 y 19, la instrucción AAM produce el mismo efecto sobre AL y AH que la AAA, pero es bastante más lenta (83 ciclos de reloj para la AAM, y sólo cuatro para la AAA), de manera que no es demasiado recomendable utilizarla. Segundo, el código máquina para la AAM:

11010100 00001010

tiene el segundo octeto igual a 10. El 8086/8088 permite colocar otros números en tal lugar, posibilitando la existencia de operaciones de ajuste ASCII para otras bases que no sean el 10.

La instrucción de ajuste ASCII para la división multiplica por 10 el contenido del registro AH, la suma a AL y borra AH. No discutiremos más esta instrucción debido a lo complejo que es la división multidígito.

Gestión de cadenas

Recordemos que una *cadena* es una secuencia de octetos o palabras. El 8086/8088 trabaja tanto en cadenas de octetos como de palabras. Técnicamente, una cadena es un tipo de dato que se guarda en un tipo de almacenamiento llamado *bloque* (véase capítulo 2). Sin embargo, la distinción entre *tipos de datos* y *tipos de almacenamiento* es a veces confusa, y especialmente en este caso. Por ejemplo, se puede hablar igualmente de mover una cadena *(movimiento de cadena)*, como de mover el contenido de un bloque de memoria *(transferencia de bloques)*.

Las operaciones con cadenas del 8086/8088 incluyen:

REP	Repetir
MOVC	Mover los caracteres de una cadena (octetos)
MOVW	Mover los caracteres de una cadena (palabras)
CMPC	Comparar caracteres de una cadena (octetos)
CMPW	Comparar caracteres de una cadena (palabras)
SCAC	Buscar caracteres en una cadena
SCAW	Buscar palabras en una cadena
LODC	Introducir caracteres en una cadena
LODW	Introducir palabras en una cadena
STOC	Guardar caracteres de una cadena
STOW	Guardar palabras de una cadena
CLD	Borrar el indicador de dirección
STD	Poner a 1 el indicador de dirección

Estas instrucciones de tratamiento de cadenas resultan útiles en las transferencias rápidas de bloques de memoria, en la búsqueda en tablas o en textos y en la codificación de datos. La idea básica de las operaciones con cadenas es similar a la del acceso directo a memoria (DMA). Esto es, a través de una pequeña secuencia de órdenes de inicialización, se establecen valores para ciertos parámetros. A continuación una nueva orden se encarga de transferir el bloque de información. Hay una diferencia, y es que las operaciones con cadenas del 8086/8088 sólo permiten movimientos de bloques dentro de la *memoria central* de computador, y *no* tienen acceso al espacio de E/S.

La instrucción REP facilita las *transferencias de bloques,* las *búsquedas en cadenas* y las *comparaciones*. Hemos hablado ya de las transferencias de bloques. Una transferencia de bloques (o movimiento de cadenas) es simplemente una instrucción que mueve todo el contenido de un bloque completo de memoria de una sola vez. Las instrucciones de cadena localizan un cierto octeto o una cierta palabra dentro de la cadena y, finalmente, las de comparación miran si dos cadenas coinciden octeto a octeto o palabra a palabra.

La instrucción REP en realidad es un octeto prefijo que se utiliza para modificar una instrucción de tratamiento de cadenas. Sólo modifica una instrucción, que debe ser como ya hemos dicho de tratamiento de cadenas, y aparecer justo a continuación del prefijo REP. El prefijo hace que la operación que le sigue se repita tantas veces como indique el registro CX (contador de repetición), dependiendo de algunas condiciones. Por ejemplo, una operación de búsqueda a la que se le haya añadido el prefijo REP para o bien cuando encuentre el/los caracteres buscados, o bien cuando el contador acabe; la condición que se cumpla primero. Existen dos versiones de la REP: una correspondiente a los nemotécnicos REP (repetición), REP (repetir mientras sea cero) y REPE (repetir mientras sea igual); y la otra correspondiente a los nemotécnicos REPNZ (repetir mientras sea distinto de 0) y REPNE (repetir mientras no sea igual). Ambas versiones trabajan igual salvo cuando modifican las instrucciones de comparación (CMPC, CMPW, SCAC y SCAW). Con los prefijos REP, REPZ o REPE, estas instrucciones se repiten o bien hasta que el contador acaba, o hasta que se encuentran valores iguales; mientras que con REPNZ y REPNE, las repeticiones acaban cuando no se encuentran valores iguales. La sintaxis para el prefijo REP es bien simple. Con el ensamblador-cruzado Microsoft, REP puede aparecer en una línea del código ensamblador, y la operación de tratamiento de cadenas subsiguiente aparece en la línea siguiente.

Las operaciones de cadenas suponen que los registros índice, el registro índice fuente (SI) y el registro índice de destino (DI) están convenientemente inicializados, SI apuntando a la fuente y DI a la posición de destino. Las operaciones de cadenas incrementan o decrementan automáticamente estos registros índice para que apunten siempre al siguiente octeto o palabra en

memoria. El indicador de dirección (DP) se usa para controlar la dirección de movimiento. Las instrucciones CLD y STD permiten al programador modificar el valor de dicho indicador. El registro de segmento de datos se utiliza normalmente con el registro índice fuente, aunque puede ignorarse. Con el registro índice de destino *siempre* se utiliza el registro de segmentación extra (ES).

Las instrucciones MOVC (mover caracteres de una cadena en la modalidad de octetos) y MOVW (para la modalidad de palabra) sirven para realizar las transferencias de bloques (junto al prefijo REP). Los datos se llevan de la posición fuente, dada por el registro índice fuente, a la posición de destino, dada por el registro índice de destino, y los índices de fuente y destino se actualizan para que apunten a las direcciones siguientes. A continuación se muestra un ejemplo de uso de dicha instrucción para la transferencia de un bloque de datos:

```
; Rutina de transferencia del bloque inicial en OLDLOC a la
nueva posición NEWLOC
; El bloque tiene 1024 octetos
;
            MOVI   SI,OLDLOC    ; la fuente está en OLDLOC
            MOVI   DI,NEWLOC    ; la fuente de destino
                                ; es NEWLOC
            MOVI   CX,1024      ; contador igual a 1024
                                ; (tamaño del bloque)
            REP                 ; prefijo de repetición
            MOVC                ; transfiere el bloque
                                ; octeto a octeto
```

Las instrucciones CMPC (modalidad de octeto), y CMPW (modalidad de palabra), comparan dos cadenas distintas, y activan o desactivan ciertos indicadores dependiendo de la diferencia entre la cadena fuente y la destino (destino menos fuente); sin modificar nunca tales cadenas. Ya hemos dicho cómo el prefijo REP hace que una instrucción se repita hasta que el contador de repeticiones se hace cero, o hasta que se detecta una igualdad (REPZ o REPE), o una desigualdad (REPNZ o REPNE). Ahora veremos un ejemplo de cómo puede verificar esta instrucción si dos bloques contienen los mismos datos:

```
; Parte de una rutina de comparación de los bloques OLDLOC y
NEWLOC
; Ambos bloques tienen 1024 octetos
;
            MOVI   SI,OLDLOC    ; la fuente está en OLDLOC
            MOVI   DI,NEWLOC    ; el puesto de destino
                                ; es NEWLOC
            MOVI   CX,1024      ; contador igual a 1024
                                ; (tamaño de los bloques)
            REPNZ               ; prefijo REP
            CMPC                ; compara los bloques
                                ; si CX=0, los bloques coinciden
                                ; en caso contrario,
                                ; son distintos
```

Las instrucciones SCAC (modalidad de octetos) y SCAW (modalidad de palabra) buscan un carácter (octeto o palabra) en una cadena (apuntada por el registro índice destino) que coincide con el valor del acumulador (AL en la modalidad de octetos, o AX en la de palabras). Según el resultado de comparación entre los elementos de la cadena y el acumulador, activan o desactivan ciertos indicadores, sin modificar nunca ni la cadena ni el acumulador. Se pueden utilizar también junto al prefijo REP de manera similar a las de comparación. En el ejemplo siguiente, se usa esta instrucción para saltar espacios en una línea de órdenes:

```
; Rutina de saldo de espacios
; DI apunta a la posición en curso del texto
;
                MOVBI  AL,2ØH   ; código ASCII del espacio
                MOV    CX,25Ø   ; tamaño máximo del buffer
                REPNZ           ; repetir hasta que no haya
                                ; un espacio
                SCAC            ; busca los espacios
                                ; DI apunta al primer carácter
                                ; que no sea un espacio
```

Las instrucciones LODC (modalidad de octeto) y LDW (modalidad de palabra) sirven para cargar un elemento de una cadena en memoria (apuntado por el registro índice fuente), en el acumulador. Tales instrucciones normalmente no se repiten. Veamos un ejemplo:

```
LODC   ; el carácter de la posición (SI)
       ; se carga en AL
       ; SI apunta el siguiente octeto
```

Finalmente, las instrucciones STOC (modalidad de octeto) y STOW (modalidad de palabra) guardan el contenido del acumulador en la posición de memoria indicada por el registro índice de destino. Normalmente tampoco estas instrucciones se repiten. Ejemplo de utilización de STOC:

```
STOC   ; guarda el contenido de AL en (DI)
       ; DI apunta al siguiente octeto
```

Las operaciones de carga (LOD) y almacenamiento (STO) son muy útiles al permitir introducir y extraer datos directamente a o del registro acumulador; registro fundamental en las operaciones aritméticas y lógicas.

Control del programa

El término *control del programa* se refiere a la tarea de controlar el *flujo* de instrucciones de un programa. Este control se consigue a través del puntero de instrucciones (IP) y el registro de segmento de código (CS). Por ejemplo, se puede saltar a una instrucción dada sencillamente cargando en registro IP (y tal vez en el CS) la dirección del «objetivo».

Las operaciones de control de programa del 8086/8088 incluyen las siguientes:

JMP	objetivo	Salto directo en el mismo segmento
JMP	objetivo segmento	Salto directo a un nuevo segmento
JMPS	destino	Salto corto
JMPI	destino	Salto indirecto en el mismo segmento
JMPL	destino	Salto indirecto largo (nuevo segmento)
JE	objetivo	Salto si es igual
JZ	objetivo	Salto si es cero
JNE	objetivo	Salto si no es igual
JNZ	objetivo	Salto si no es cero
JS	objetivo	Salto si es negativo
JNS	objetivo	Salto si es no-negativo
JP	objetivo	Salto si hay paridad (par)
JNP	objetivo	Salto si no hay paridad (paridad impar)
JPE	objetivo	Salto si hay paridad par
JPO	objetivo	Salto si hay paridad impar
JL	objetivo	Salto si es menor que
JNGE	objetivo	Salto si no es mayor o igual a
JNL	objetivo	Salto si no es menor que
JGE	objetivo	Salto si es mayor o igual que
JLE	objetivo	Salto si es menor o igual que
JNG	objetivo	Salto si no es mayor que
JNLE	objetivo	Salto si no es menor o igual a
JG	objetivo	Salto si es mayor que
JB	objetivo	Salto si está por debajo
JNAE	objetivo	Salto si no está por encima o es igual a
JNB	objetivo	Salto si no está por debajo
JAE	objetivo	Salto si está por encima o es igual a
JBE	objetivo	Salto si está por debajo o es igual a
JNA	objetivo	Salto si no está por encima
JNBE	objetivo	Salto si no está por debajo o es igual a

TEST	destino, fuente	Verifica
TESTB	destino, fuente	Verifica octeto
TESTI	destino, dato	Verifica dato inmediato
TESTBI	destino, dato	Verifica octeto con dato inmediato
CMP	destino, fuente	Compara
CMPB	destino, fuente	Compara octeto
CMPI	destino, dato	Compara con dato inmediato
CMPBI	destino, dato	Compara octeto con dato inmediato
LOOP	objetivo	Bucle
LOOPZ	objetivo	Bucle si es cero
LOOPNZ	objetivo	Bucle si no es cero
LOOPE	objetivo	Bucle si es igual
LOOPNE	objetivo	Bucle si no es igual
JCXZ	objetivo	Salta si CX es cero
CALL	objetivo	Llamada directa en el mismo segmento
CALL	objetivo, segmento	Llamada directa a nuevo segmento
CALLI	destino	Llamada indirecta en el mismo segmento
CALLL	destino	Llamada indirecta larga (nuevo segmento)
RET		Vuelta en el mismo segmento
RET	número	Vuelta y ajuste de pila
RETS		Vuelta desde el segmento
RETS	número	Vuelta desde el segmento y ajuste de pila

La instrucción de salto incondicional (JMP) para controlar a otra área del programa; como la pulga que va avanzando y de pronto salta a otro punto del perro.

La instrucción JMP admite una serie de variantes que vamos a ver a continuación.

JMP con un operando salta directamente a una dirección dada por el objetivo dentro del segmento en curso. El operando contiene en realidad una dirección relativa a la que se da el nombre de desplazamiento. La dirección real se obtiene sumándole a ésta el contenido del puntero de instrucción (IP). Hay que tener cuidado porque IP apunta siempre a la instrucción siguiente

que debe ejecutarse, y no a la instrucción que produce el salto. Afortunadamente, el ensamblador se encarga de solventar esos problemas. El lenguaje ensamblador permite utilizar rótulos para determinar la posición del objetivo, de modo que el operando de la instrucción JMP puede ser un rótulo, y no la dirección relativa. El ensamblador se encarga de calcular esta dirección relativa y de introducirla en el código máquina. Este direccionamiento relativo es muy útil si queremos trabajar con un *código independiente de la posición;* esto es, con un código que no necesite modificación si se traslada como un todo a otra posición. Dicho de otra manera, si el código es *reubicable*. Este formato del JMP reserva 16 bits para la dirección relativa (diferencia entre la dirección del objetivo y el contenido de IP). Aquí presentamos un ejemplo:

```
JMP   FUN1      ; salta a la función #1
                ; en este mismo segmento
```

La instrucción JMP con dos operandos salta directamente a una dirección determinada por el objetivo que puede estar en cualquier segmento. El desplazamiento dentro del segmento viene dado por el primer operando, y el contenido a cargar en el registro de segmentación (segmento destino) viene dado por el segundo operando. En este caso el direccionamiento relativo no resulta apropiado al llegar a un segmento distinto al de partida. El contenido de registro de segmento de código aprovecha las ventajas de la reubicabilidad, y el desplazamiento del objetivo es relativo sólo respecto al punto de comienzo del segmento. Veamos un ejemplo:

```
JMP   FUN1,SYS1   ; salta a la función #1
                  ; en el segmento SYS1
```

La instrucción JMPS es una versión acortada de la instrucción JMP, con un operando. JMPS salta directamente al objetivo, dentro del mismo segmento, usando direccionamiento relativo de 8 bits para el desplazamiento. Así, por ejemplo:

```
JMPS   FUN1    ; FUN1 debe estar en el mismo
               ; segmento y cerca
```

JMPI salta indirectamente dentro del mismo segmento. Indirectamente significa que el operando (referenciado por cualquiera de los 25 modos de direccionamiento) contiene la dirección del desplazamiento de la posición objetivo. Es una extensión bastante sofisticada de la instrucción PCHL del 8080/8085, que intercambiaba los contenidos del contador de programa y del registro HL del 8080/8085. La instrucciön JMPI del 8086/8088 permite introducir en el puntero de instrucción (IP) el contenido de cualquier posición de memoria. Por ejemplo:

```
JMPI   BX   ; equivalente a PCHL
            ; del 880
```

La instrucción JMPL produce un salto indirecto a cualquier segmento. El operando se puede referenciar por cualquiera de los 25 modos de direccionamiento. En este caso, el contenido del operando se coloca en el IP, y los dos octetos siguientes se introducen en el registro de segmento de código. Veamos un ejemplo:

```
JMPL    [BX]    ; BX apunta a las
                ; 2 palabras de dirección
                ; del objetivo
```

El 8086/8088 posee un juego completo de saltos condicionales dependientes de una gran variedad de condiciones, desde el estado de ciertos indicadores, a relaciones aritméticas o relaciones entre valores sin signo. La tabla 3.3 muestra un resumen de todas ellas.

Condiciones sobre indicadores	
JE	Salto si es igual
JZ	Salto si es cero
JNE	Salto si no es igual
JNZ	Salto si no es cero
JS	Salto si es negativo
JNS	Salto si es positivo o nulo
JP	Salto si hay paridad
JNP	Salto si no hay paridad
JPE	Salto si hay paridad par
JPO	Salto si hay paridad impar
Relaciones aritméticas con signo	
JL	Salto si es menor que
JNGE	Salto si no es mayor o igual que
JNL	Salto si no es menor que
JGE	Salto si es mayor o igual que
JLE	Salto si es menor o igual que
JNG	Salto si no es mayor que
JNLE	Salto si no es menor o igual que
JG	Salto si es mayor que
Relaciones aritméticas sin código	
JB	Salto si está por debajo
JNAE	Salto si no está por encima o es igual a
JNB	Salto si no está por debajo
JAE	Salto si está por encima o es igual a
JBE	Salto si está por debajo o es igual a
JNA	Salto si no está por encima
JNBE	Salto si no está por debajo o es igual a

Tabla 3.3: Saltos condicionales.

Los saltos condicionales usan el mismo direccionamiento directo relativo de 8 bits que la instrucción JMPS. Observando el código máquina de estos saltos condicionales (véase apéndice D), puede verse que hay varias instrucciones con el mismo código máquina. Por ejemplo, JE, JZ, JNE y JNZ.

Las instrucciones de verificación y comparación se utilizan normalmente para establecer condiciones para los saltos condicionales. Todas estas instrucciones tienen dos operandos, fuente destino, a los que nunca modifican. Se limitan a activar o desactivar indicadores. Las de verificación activan los indicadores dependiendo del producto lógico (AND) de los dos operandos, mientras que las de comparación los activan dependiendo del resultado obtenido al restar la fuente al destino.

Veamos con más detalle cómo funcionan las instrucciones de comparación en los saltos condicionales.

Cuando la configuración de los indicadores generada por la instrucción de comparación sirva para determinar si se realiza o no el salto condicional, la instrucción CMP debe preceder inmediatamente a la de salto, debido a que si hubiese entre ellas cualquier otra instrucción, se podría modificar erróneamente el valor de los indicadores. Debemos tener siempre la siguiente secuencia de instrucciones.

```
CMP                  destino, fuente     ; compara
salto condicional    objetivo            ; salto condicionado al
                                         ; resultado de comparación
```

donde «salto condicional» puede ser cualquiera de las instrucciones de la tabla anterior. La secuencia debería leerse así: «SI», léanse los operandos de CHP reemplazando la coma por la condición, «salto a», finalmente se lee el objetivo (el operando del salto condicional). Por ejemplo:

```
CMP   BX,DX    ; si BX es mayor que DX
JGE   GETIT    ; salta a GETIT
```

Es interesante resaltar el hecho de que las instrucciones CMP de 8086/8088 siguen el mismo orden de substracción que la operación de resta normal. En el PDP-11/LST-11, se sigue el orden inverso, debido a que el PDP-11/LST-11 sigue el orden inverso en las operaciones de dos operandos, con la fuente precediendo a destino. Para que se lea correctamente una secuencia de comparación/salto condicional en el PDP-11, es necesario invertir el orden de la resta.

La instrucción LOOP se usa para controlar la ejecución de bucles. Se coloca al final del bucle, y básicamente es equivalente a:

```
DEC   contador                ; decrementa el contador y
                              ; vuelve al comienzo del
JNZ   comienzo-de-bucle       ; bucle si no es cero
```

se decrementa en uno el valor de un cierto contador, y se repite el bucle mientras el contador no llegue a cero. Las diferencias entre esta secuencia y la instrucción LOOP son las siguientes: 1) la instrucción LOOP no modifica los indicadores, mientras que la DEC sí puede hacerlo, y 2) la instrucción LOOP obliga a que el contador esté en el registro CX (registro contador).

Hay en realidad tres versiones de la instrucción LOOP: LOOP, LOOPZ/LOOPE y LOOPNZ/LOOPNE.

LOOP funciona de la siguiente manera: decrementa el registro CX, y si CX es distinto de cero, salta al objetivo. No se modifica ningún indicador. Así por ejemplo:

```
        MOVI   CX,24   ; contador = 24
START:  LODC           ; obtener el octeto
        ANDI   7FH     ; aislar el bit de paridad
        STOC           ; guardar el octeto
        LOOP   START   ; repetir el bucle hasta
                       ; que el contador sea 0
```

Las instrucciones LOOPZ y LOOPE funcionan de la siguiente manera: Se decrementa el registro CX y si el indicador de cero (ZF) toma el valor 1 y CX no es cero, se salta al objetivo. No se modifica ningún indicador.

LOOPNZ y LOOPNE hacen lo siguiente: se decrementa el registro CX, y si ZF es cero y CX no lo es, se salta al objetivo. De nuevo no se modifica ningún indicador. Por ejemplo:

```
START:  LODC            ; obtener el octeto
        ANDI    7FH     ; aislar el bit de paridad
        STOC            ; guardar el octeto
        LOOPNZ  START   ; repetir el bucle mientras
                        ; el contador sea distinto de 0
                        ; y el octeto sea distinto de 0
```

La instrucción JCXZ está muy relacionada con la instrucción LOOP. Trabaja de la siguiente manera: si CX es 0, se salta al objetivo. Es muy útil al comienzo del bucle para asegurarse de que éste no se va a ejecutar si el contador inicialmente es 0. Es útil también combinada con las de búsquedas repetidas (REP, SCA), y las de comparación repetidas (REP, CMP), para realizar saltos condicionados a que dos cadenas dadas coincidan. Es decir, si REP SCAC, REP SCAW, REP CMPC o REP CMPW, se ha encontrado que no ha habido coincidencia antes de que el contador llegara a cero, entonces CX será 0, y la instrucción JCXZ generará un salto. Si por el contrario se ha dado una igualdad, antes de que el contador llegara a cero, JCXZ no provocará ningún salto.

Las instrucciones CALL y RET controlan la ejecución de subrutinas. El 8086/8088 guarda en la pila del sistema (a través del puntero de pila) las direcciones de vuelta de las subrutinas. Cada vez que llama a una subrutina (CALL), la dirección de

vuelta se introduce en la pila, y cada vez que se acaba una subrutina (RET), la dirección de vuelta se extrae de la pila.

La instrucción CALL tiene prácticamente las mismas versiones que la de salto incondicional, salvo la versión corta (JMPS).

Para llamar a subrutinas que estén en el mismo registro se utiliza la instrucción CALL con un operando. Esta instrucción sencillamente provoca un salto directo, utilizando un desplazamiento *relativo* de 16 bits para calcular el desplazamiento real dentro del segmento. Por ejemplo:

```
CALL  ADDIT   ; en el mismo segmento
```

La versión con dos operandos se utiliza para llamadas a subrutinas en cualquier segmento. El primer operando es el desplazamiento y el segundo es el nuevo contenido del registro de segmentación. Como en la JMP a otro segmento, el desplazamiento es *absoluto* y *no* relativo. Por ejemplo:

```
CALL  CDOUT,IOSEC  ; en segmentos distintos
```

CALLI y CALLL son llamadas indirectas, CALLI para subrutinas en el mismo segmento, y CALLL para subrutinas en cualquier segmento. En el primer caso, el operando puede referenciarse por cualquiera de los 25 modos de direccionamiento posibles. Este operando contiene el desplazamiento. En el segundo caso, el operando se puede referenciar por cualquiera de los 24 modos de referencia a memoria. Contiene el desplazamiento, mientras que la palabra siguiente contiene la dirección del nuevo segmento. Veamos un ejemplo:

```
CALLI  AX   ; AX contiene la dirección
```

La instrucción RET sirve para volver de una subrutina en el mismo segmento, y RETS permite que la subrutina esté en cualquier segmento. Puesto que la instrucción RET forma parte del cuerpo de la subrutina (es la última instrucción), la instrucción RET o RETS define la subrutina como *local* (cuando está en el *mismo* segmento) o *global* (debe llamarse con llamadas «largas»). Así, por ejemplo:

```
RET   ; vuelta al programa principal

RETS  ; vuelta al programa principal
      ; que está en otro segmento
```

Ambos tipos de RET pueden o bien no tener operandos o tener uno. La división con operando sirve de ayuda a la hora de pasar datos a o desde la subrutina llamada al procedimiento de llamada. El operando se suma al puntero de pila después de que

se extraiga la dirección de vuelta, con lo cual se salta cualquier dato que haya sido introducido en la pila, sin extraerlo. Por ejemplo:

```
RET 2    ; vuelve y salta 2 octetos de datos
```

Control de sistema

Se incluyen aquí una serie de instrucción bajo el nombre de *control de sistema* que incluyen instrucciones de interrupción software, la instrucción de parada (HLT) y espera (WAIT) e instrucción de gestión de los indicadores.

Las operaciones de control del sistema del 8086/8088 incluyen las siguientes:

INT	Interrupción
INTO	Interrupción si hay capacidad excedida
IRET	Vuelta de interrupción
CLI	Borrar el indicador de interrupción
STI	Activar el indicador de interrupción
HLT	Parada
WAIT	Espera
LOCK	Bloques
ESC	Escape
NOP	No operación
CLC	Borra el arrastre
STC	Pon a 1 el arrastre
CMC	Complementa el arrastre
SAHF	Guarda AH en los indicadores
LAHF	Carga AH con el valor de los indicadores
PUSHF	Introduce los indicadores (en la pila)
POPF	Extrae los indicadores (de la pila)

La instrucción INT provoca interrupciones software. Tiene un operando que especifica el «tipo» de interrupción (revísese la estructura de interrupciones del 8086/8088 (8086/8088 ya explicada). La instrucción INTO produce una interrupción «tipo 4» si el indicador de capacidad excedida (OF) está a 1, y resulta útil en operaciones como la división. La instrucción IRET devuelve el control al programa de llamada a la rutina de interrupción, INT, es algo distinta que las llamadas a subrutinas (CALL) (de hecho INT introduce más información en la pila). De la misma manera, la instrucción IRET es también específica de las rutinas de procedimiento de las interrupciones, de manera que no puede sustituirse por una RET. Las instrucciones CLI y STI desactivan y activan respectivamente las interrupciones. El capítulo 7 incluye un ejemplo de estas instrucciones de interrupción software.

La instrucción HLT para el funcionamiento de la CPU. La CPU se puede volver a poner en funcionamiento o bien mediante una interrupción externa, o mediante un «reset» (borrado) general. Resulta útil cuando la CPU acaba su trabajo y necesita nuevas entradas antes de continuar.

Las instrucciones ESC y WAIT enlazan la CPU con un procesador paralelo.

ESC pone en funcionamiento el procesador paralelo. Tiene un solo operando que puede utilizar cualquiera de los 25 modos de direccionamiento, y que es utilizado en realidad por el procesador paralelo en vez de por la CPU. En el capítulo 4 dedicado al NDP 8087 se trata el tema de los procesadores paralelos con más detalle.

La instrucción WAIT se emplea para sincronizar la CPU y el procesador paralelo (por ejemplo, el NDP 8087). La instrucción WAIT hace que la CPU pare hasta que le entre una señal por el terminal TEST indicándole que continúe. El terminal $\overline{\text{TEST}}$ de la CPU se conecta en estos casos al terminal BUSY del procesador paralelo. De nuevo en el capítulo 4 se dan más detalles.

LOCK es en realidad un prefijo. Su misión es prevenir conflictos en el contexto de la multiplicación. Evita que el bus pueda ser utilizado por más de un procesador durante la instrucción siguiente a ella misma. Los esquemas de protección software (por medio del octeto de ocupación) tienen su talón de Aquiles: dos procesos pueden intentar acceder simultáneamente al octeto de ocupación. Este caso se evita con la instrucción LOCK. El capítulo dedicado al IOP 8089 trata el tema con más detalle.

La instrucción NOP no hace absolutamente nada, salvo emplear tiempo y espacio. Es equivalente a intercambiar AX consigo mismo, pero contra lo que pueda parecer, este gasto aparentemente inútil de tiempo y espacio puede ser útil en ciertas circunstancias. Por ejemplo, las rutinas E/S en programas código máquina suelen acabar con instrucción NOP que, al reservar más espacio, permiten al usuario definir rutinas de E/S de mayor tamaño que las originales. La operación NOP puede ser también útil para resolver pequeños problemas de sincronización.

Las instrucciones CLC, STC y CMC permiten activar o desactivar directamente el indicador de arrastre.

Las SAHF y LAHF permiten acceder a los indicadores ZF, AF y PF, localizados en el octeto inferior del registro de indicadores. Veamos algún ejemplo de su uso:

```
; lo que sigue equivale a un PUSH PSW
; del 8080
        LAHF          ; llevar indicadores a AH
        XCHG   AH,AL  ; para contabilizarlo con el 8080
        PUSH   AX     ; introducir AL y los indicadores

; lo que sigue equivale a una POP PSW
; del 8080
        POP    AX     ; extraer AX y los indicadores
        XCHG   AH,AL  ; para contabilizarlo con el 8080
        SAHF          ; llevar AH a los indicadores
```

Para obtener el conjunto completo de indicadores se necesitan las instrucciones PUSHF (introducir indicadores en la pila) y POPF (extraer indicadores de la pila). Así, por ejemplo:

```
PUSHF        ; introducir indicadores en la pila
POP    AX    ; ahora se encuentran en AX
```

Comparación con otros procesadores

Miremos el juego de instrucciones de 8086/8088 desde otro ángulo, su relación entre este juego de instrucciones, el de sus predecesores, los microprocesadores de 8 bits y el de sus rivales de 16 bits.

El juego de instrucciones de 8086/8088 es mucho más completo que los de sus predecesores directos, el 8080/8085. Por ejemplo, aparte de extender el juego de instrucciones para poder gestionar datos de 16 bits, tienen nuevas instrucciones como: la multiplicación y división entera, un conjunto completo de instrucciones de tratamiento de cadenas y operaciones completamente nuevas como la LOCK y ESCP para el control del multi-proceso y del procesamiento en paralelo.

Cualquier comparación que se haga con sus rivales de 16 bits como el Motorola MC68000 y el Zilog Z8000 ha de tener en cuenta el hecho de que tanto los procesadores Zilog como Motorola son en realidad máquinas de 32 bits encapsuladas en paquetes de 8 bits. El MC68000 y él tienen en realidad instrucciones de 32 bits (además de instrucciones de 8 y 16 bits), pero un bus de datos de 16 bits. Es una situación similar a la del 8088, que internamente trabaja con 16 bits, pero su bus de datos externo es de 8. Por eso el 8088 se ha convertido en líder en el mercado de los procesadores de 8 bits.

La comparacióndel juego de instrucciones de Intel, Motorola y Zilog con respecto a las nueve categorías en las que se han dividido las instrucciones muestra al 8086/8088 como un procesador muy adecuado salvo, quizás, en el grupo de tratamiento de bits; pero incluso aquí, con el uso de «macros» se pueden obtener resultados razonablemente buenos.

Respecto a la velocidad de proceso, el Motorola suele ganar fácilmente. Sin embargo, en la mayoría de operaciones estándar aritmético/lógicas tales como AND o ADD, el 8086/8088 necesita menos ciclos de reloj que el Z8000 o el MC68000. Por otro lado, puesto que el MC68000 se diseñó para funcionar con un sistema de reloj doble, con una velocidad mucho mayor para el procesador que para cualquiera de los periféricos, se puede construir un sistema MC68000 con memoria razonablemente lenta que ejecute una suma (ADD) registro-a-registro (sin tocar la memoria) en un tiempo menor que un sistema basado en el 8086, de reloj simple. Este truco puede utilizarse con cualquier procesador, pero parece que el sistema más correcto sería aumentar la velocidad de la memoria. La tabla 3.4 compara el número de ciclos que necesitan estos tres procesadores en algunos casos típicos de la instrucción ADD.

	(Ciclos			(Tiempo en microsegundos)					
	8086	Z8000	MC68000	8086		Z8000		MC68000	
				5 mhz	8 mhz	4 mhz	10 mhz	4 mhz	10 mhz
Registro-a-registro	3	4	4	0,6	0,375	1,00	0,4	1,00	0,4
Registro a dirección directa	22	—	21	4,4	2,750	—	—	5,25	2,1
Dirección directa a registro	15	9	16	3,0	1,875	2,25	0,9	4,00	1,6
Dato inmediato a registro	4	7	8	0,8	0,500	1,75	0,7	2,00	0,8
Dato inmediato a dirección directa	23	—	25	4,6	2,875	—	—	6,25	2,5

Nota: Significa que esta variación particular no es posible.

Tabla 3.4: Comparación de velocidades para la suma.

La posibilidad de disponer de unas operaciones potentes de multiplicación y división son una buena razón para ampliar el sistema de 8 bits a 16. El 8086/8088 posee instrucciones de multiplicar y dividir para números con o sin signo, de 8 ó 16 bits, mientras que el Zilog tiene *sólo* la multiplicación para números con signo; y ni el MC68000 ni el 78000 poseen versiones de 8 bits para la multiplicación o división. Sin embargo, las versiones de 8 bits de la multiplicación en el Z8000 y el MC68000 tienen una velocidad comparable a las versiones de 8 bits del 8086; y el 8086 es definitivamente más lento en la multiplicación y división de números de 16 bits. Intel planea mejorar estos detalles en el 186. Las tablas siguientes dan una comparación entre las operaciones de multiplicar y dividir.

Las instrucciones de tratamiento en cadenas del 8086/8088 son bastante completas. El Z8000 posee un conjunto completo de operaciones de éstas, mientras que el MC68000 utiliza instrucción estándar con modos de direccionamiento más elaborados consiguiendo prácticamente resultados idénticos. Por lo general, el Z8000 es el que necesita menos ciclos de reloj en este tipo de operaciones (incluida la repetición), y el MC68000 es el que necesita más. Por ejemplo,una búsqueda con repetición en el 8086/8088 (REP SCAS) necesita 9+15n ciclos de reloj (n es el número de repeticiones), mientras que la operación equivalente de Z8000 (CPDR: Compara, Decrementa y Repite) necesita 11+9n ciclos. El MC68000 puede realizar la misma operación con una instrucción de comparación (CMP) con modo de direccionamiento «indirecto con predecremento» seguida de una instrucción de bucle (DBNE). Para hacer esto necesitaría unos

	(Ciclos) 8086	Z8000	MC68000	(Tiempo en microsegundos) 8086 5 mhz	8086 8 mhz	Z8000 4 mhz	Z8000 10 mhz	MC68000 4 mhz	MC68000 10 mhz
8 bits con signo									
Registro-a-registro	90	—	—	18,0	11,250	—	—	—	—
Dirección directa a registro	96	—	—	19,2	12,000	—	—	—	—
16 bits con signo									
Registro-a-registro	144	70	70	28,8	18,000	17,50	7,4	17,50	—
Dirección directa a registro	150	71	78	30,0	18,750	17,75	7,1	19,50	—
8 bits sin signo									
Registro-a-registro	71	—	—	14,2	8,875	—	—	—	—
Dirección directa a registro	77	—	—	15,4	9,625	—	—	—	—
16 bits sin signo									
Registro-a-registro	124	—	70	24,8	15,500	—	—	31,00	7,0
Dirección directa a registro	130	—	78	26,0	16,250	—	—	32,50	7,8

Nota: Significa que esta variación particular no es posible.

Tabla 3.5: Comparación de velocidades en la multiplicación.

	(Ciclos) 8086	Z8000	MC68000	(Tiempo en microsegundos) 8086 5 mhz	8086 8 mhz	Z8000 4 mhz	Z8000 10 mhz	MC68000 4 mhz	MC68000 10 mhz
8 bits con signo									
Registro-a-registro	112	—	—	22,4	14,000	—	—	—	—
Dirección directa a registro	118	—	—	23,6	14,750	—	—	—	—
16 bits con signo									
Registro-a-registro	177	95	158	35,4	22,125	23,75	9,5	39,50	15,8
Dirección directa a registro	183	96	166	36,0	22,875	24,00	9,6	41,50	16,6
8 bits sin signo									
Registro-a-registro	90	—	—	18,0	11,250	—	—	—	—
Dirección directa a registro	96	—	—	19,2	12,000	—	—	—	—
16 bits sin signo									
Registro-a-registro	155	—	140	31,0	19,375	—	—	35,00	14,0
Dirección directa a registro	161	—	148	32,2	20,125	—	—	37,00	14,8

Nota: Significa que esta variación particular no es posible.

Tabla 3.6: Comparación de velocidades en la división.

2+18n ciclos de reloj. Como puede observarse en la figura 3.18, la función 2+18n crece más rápidamente que la 19+15n (8086) y 11+9n (Z8000), de modo que para búsquedas múltiples, el MC68000 resulta más lento.

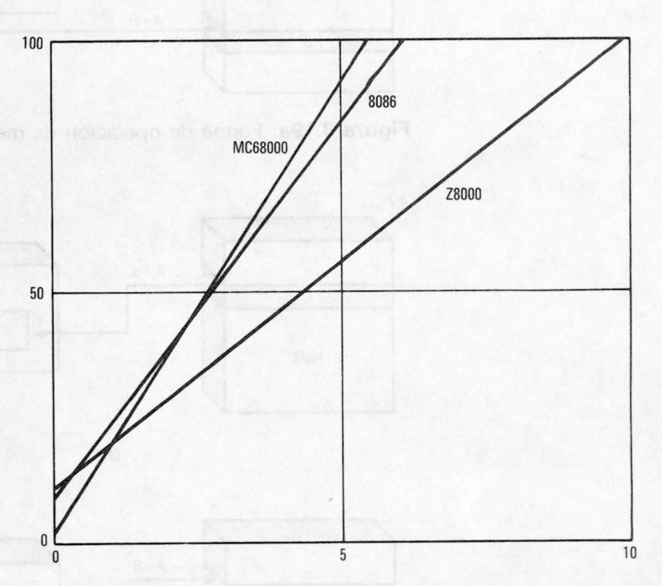


Figura 3.18: Comparación de velocidades para búsquedas múltiples.

La mayoría de las instrucciones del 8086/8088 no permiten el movimiento de datos directo de memoria a memoria, cosa muy común en mini y maxi-computadoras. El dato debe llevarse primero a un registro de la CPU (véase la figura 3.19). Sin embargo, las operaciones de tratamiento de cadenas permiten los movimientos memoria-a-memoria, y hay otros tipos de instrucciones que permiten el llevar a memoria datos inmediatos. (Un dato inmediato es un dato que es parte del programa.) Los otros dos procesadores tienen el mismo problema. Por ejemplo, con la instrucción ADD, tanto el 8086/8088 como el MC68000 pueden sumar de registro a registro, de registro a memoria, de memoria a registro, un dato inmediato a registro y un dato inmediato a memoria; mientras que el Z8000 sólo puede hacerlo de registro a registro, memoria a registro o dato inmediato a registro.

En una minicomputadora como el PDP-11 (y en el micro LSI-11), se puede realmente sumar de memoria a memoria, cosa muy útil. Se dice que muchos procesadores de 16 bits están inspirados en el PDP-11 de Digital Equipment, aunque también es cierto que ninguno de ellos ha conseguido la potencia de direccionamiento del PDP. Es de resaltar la similitud entre los sistemas operativos del PDP-11 y los programas de ayuda del CP/M, que es el sistema operativo más estándar hoy en día para el 8080, 8085, Z80, 8086 y 8088 (véase Murtha, Stephen M.;

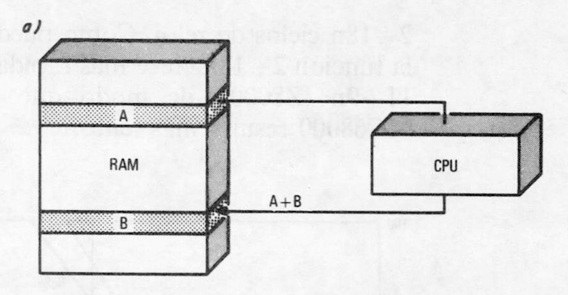


Figura 3.19a: Forma de operación de memoria a memoria.

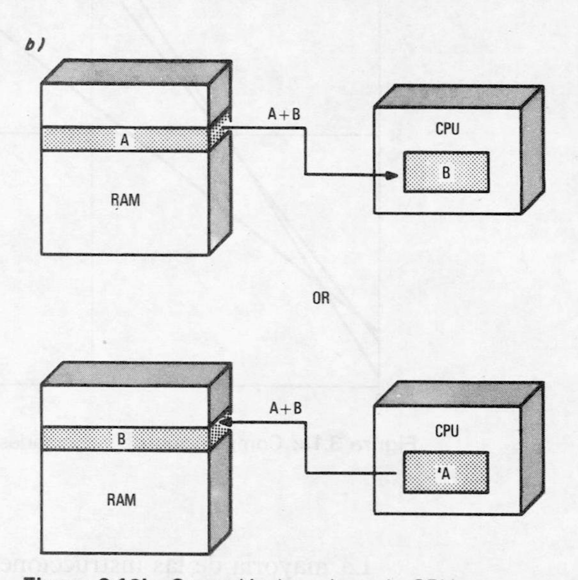


Figura 3.19b: Operación basada en la CPU.

Waite, Mitchell: *CP/M Primer*. Indianápolis, Indiana: Howard W. Sams & Co., Inc.).

La codificación de las instrucciones del 8086/8088 es por octetos. No es así en otros procesadores de 16 bits como el MC68000, Z8000 y LSI-11, en los cuales las instrucciones van de 16 en 16 bits. En el nuevo iAPX 432 de 32 bits de Intel, las instrucciones se codifican en grupos de bits de tamaño variable dependiendo de las necesidades de la instrucción. Es decir, no hay un tamaño fijo de instrucción como en las otras máquinas de 16 bits.

En conjunto y por sí mismo, el 8086/8088 parece menos potente que los otros dos procesadores estudiados: gestiona sólo datos de 8, 16 y 32 bits, e incluso tienen alguna operación disponible entre enteros de 64 bits, necesita más ciclos de reloj que el Z8000 en las operaciones de tratamiento de cadenas y tiene un espacio de direccionamiento menor (20 bits) que los otros procesadores. Y es que el 8086 y el 8088, son en realidad, una clase distinta de procesadores, llamados a ocupar áreas distintas en la curva de precios/rendimiento. En particular, el 8088 comienza a erigirse en un líder en el mercado de los microprocesadores

de 8 bits, debido a su mayor potencia (trabaja intensamente con 16 bits), y a su precio, que está llegando a niveles competitivos con los de los microprocesadores de 8 bits. Aparentemente, el 8086 es punto de partida de toda una línea de procesadores de 16 bits de alto rendimiento. El 8086 fue el primero de la nueva generación de microprocesadores de 16 bits, y por tanto, el campo de pruebas de algunas ideas nuevas. Algunas de estas ideas necesitan ahora retoques, refinamientos y a veces ampliaciones y esto es lo que Intel protende hacer con el iAPX 186 (fundamentalmente refinamientos) y el iAPX 286 (ampliaciones en el área de la gestión de memoria. ¡La batalla acaba de comenzar!

Cuando se usa el 8086/8088 junto al Procesador de Datos Numérico 8087 y el Procesador de Entrada/Salida 8089, el programa cambia radicalmente, y la potencia del 8086/8088 llega a niveles insospechados.

Con el NDP 8087, de pronto, es posible trabajar con números en coma flotante y alta precisión, ¡a más velocidades prácticamente iguales a las que requerirá la multiplicación o división de números enteros si se realizara con sólo la CPU! Por otro lado, el IOP 8089 permite realizar operaciones orientadas a cadenas (usando incluso una tabla de traducción) a altas velocidades. Por ejemplo, una transferencia de bloques utilizando una tabla de traducción y haciendo una verificación de terminación necesita 15 ciclos de reloj por octeto en el IOP 8089, y 25 en el Z8000. Además, mientras el 8089 está realizando la transferencia, ¡la CPU 8086/8088 y el NDP 8087 pueden estar haciendo otras cosas! Para cada uno de estos procesadores auxiliares se ha reservado un capítulo.

LENGUAJE MAQUINA PARA UNA INSTRUCCION TIPICA

Veamos cómo se implementa en lenguaje máquina una instrucción concreta. Tomaremos como ejemplo la instrucción ADD por funcionar exactamente igual que otras siete instrucciones. La OR, ADC, SBB, AND, SUB, XOR y CMP. Todas ellas tienen dos operandos, fuente y destino. En cada caso, los contenidos de la fuente y destino se operan según la instrucción particular y el resultado se lleva al punto de destino.

Existen actualmente varios ensambladores disponibles para el 8086/8088; entre ellos el propio ensamblador de Intel, el ASM86, y el ensamblador cruzado de Microsoft. Recordemos que un ensamblador cruzado es un ensamblador que funciona sobre una máquina, pero produce código para otra. En este caso, el ensamblador trabaja sobre un 8080, 8085 o Z80 con CP/M y produce código para el 8086 o 8088.

El ensamblador cruzado de Microsoft tiene las siguientes variantes de código para esta instrucción.

ADD	Destino, fuente	Suma fuente a destino (palabra)
ADDB	Destino, fuente	Suma fuente a destino (octeto)
ADDI	Destino, dato	Suma el dato a destino (palabra)
ADDBI	Destino, dato	Suma el dato a destino (octeto)

La I y la B adicional las usa el ensamblador de Microsoft como ayuda para indicar la información que el ensamblador Intel determina automáticamente cuando mira los datos. La I significa que la fuente es un dato inmediato; esto es, viene especificado como parte de la instrucción, y la B significa que tanto la fuente como destino son octetos (cantidades de 8 bits). La ausencia de B se interpreta como que tanto la fuente como destino son palabras (cantidades de 16 bits).

Los nemotécnicos ADD y ADDB implementan en lenguaje máquina de varias maneras dependiendo de 1) las posiciones de fuente y destino, por ejemplo, a registro de registro, a registro de memoria o a memoria de registro, 2) el tamaño de los datos, 3) el tamaño del decalage, 4) los registros de base a índice utilizados y 5) los registros de estos considerandos.

En cualquier caso, el primer octeto del código máquina es:

$$00ccc0dw$$

El primer octeto básicamente dice la operación, pero también informa sobre el tamaño de los operandos y dice de dónde están. Más concretamente, se codifica de la siguiente manera: $ccc=000$ es un código binario de 3 bits que indica que se trata de la instrucción ADD. La d es un código binario de 1 bit que indica la dirección. Si el destino está en memoria y la fuente es un registro, $d=0$; si el destino es un registro y la fuente está en memoria, $d=1$. La w indica el tamaño de los datos. Si los datos son de 8 bits (octeto) entonces $w=0$ (en este caso se usaría una *B* con el ensamblador Microsoft), y si son de 16 bits (palabra), entonces $w=1$ y no se usaría la *B*.

El segundo octeto:

$$mm\ rrr\ aaa$$

Da información sobre los operandos (incluyendo el modo de direccionamiento). Concretamente, *mm* es un código binario de 2 bits que indica parte del modo de direccionamiento. Los 3 bits *aaa* completan el resto del modo de direccionamiento. El código binario *rrr* indica un registro. Si $d=0$, *rrr* indica al registro fuente, y la otra información sobre el direccionamiento determina el destino; y si $d=1$, *rrr* indica el registro destino y el resto de la información sobre el direccionamiento determina la fuente.

Dependiendo de *mm* y *aaa*, puede haber bits adicionales que indiquen valores para el decalage de la dirección. La tabla 3.1 muestra estos detalles.

Resumiendo, consideramos el siguiente ejemplo: el código fuente ensamblador es:

```
ADD   6[BX][DI],DX    ; suma DX a la posición
                      ; DI+BX+6
```

Esta instrucción suma el contenido de DX a la posición descrita por el mismo modo de direccionamiento que el usado en

el ejemplo de modos de direccionamiento. Se define la dirección a través de una base (registro BX), un índice (DI) y un decalage de 6. Puesto que la fuente es un registro y el destino está en memoria, $d=0$. Como además DX indica un registro de 16 bits, el tamaño de los datos será de 16, y por tanto, $w=1$. El decalage es de 8 bits, y la tabla de modos de direccionamiento (tabla 3.1) podemos ver que significa que $mm=01$. En la misma tabla observamos que $aaa=001$. El código para el registro DX es $rrr=010$. Sólo se necesitará 1 bit adicional para el decalage (de 8 bits). Resumiendo todo esto tendremos:

```
ADD 6[BX][DI],DX
=> 00 000 001    01 010 001    00000110
=>       01           51        06 hex
```

En condiciones normales, el ensamblador se encarga de hacer todo este trabajo, pero resulta interesante comprender exactamente cómo, para cuándo se tenga que corregir el código.

La mayor parte de las veces los operandos no son tan complicados. Por ejemplo:

```
ADD    AX,[SI]    ; suma la posición (SI)
                  ; al registro AX
```

En este caso, el índice fuente apunta a la fuente; y el destino es el registro ZX. El código máquina correspondiente sería:

00 000 011   00 000 100

Aquí tenemos w=1 (dato tamaño palabra), d=1 (de memoria a registro) mm=00 (sin decalage), aaa=100 (modo índice, con el SI apuntados a los datos) y rrr=000 (registro AX para otro operando). Como no hay decalage, no aparecen bits adicionales.

Veamos ahora los nemotécnicos ADDI (Suma Dato Inmediato) y ADDBI (Suma Dato Inmediato: tamaño octeto). Estos nemotécnicos se traducen a código máquina de dos formas distintas. Esto es típico de la forma en que el 8086/8088 se ha diseñado para optimizar ciertas combinaciones usadas con mucha frecuencia, especialmente aquellas que utilizan registros como el acumulador (AX o AL).

La forma más general tiene como fuente un dato inmediato y como destino un registro general o una posición de memoria. El primer octeto es:

100000*sw*

Aquí, la *s* es un código binario de 1 bit que indica el tamaño del dato inmediato, y *w* es otro código binario de 1 bit que indica el tamaño del dato en destino. Si *sw* es 00, tanto la fuente como destino tienen 8 bits; si *sw* es 11, ambos datos son de 16 bits, y si *sw* es 01, indica que la fuente es de 8 bits y se debe ampliar a 16

bits (usando aritmética entera de complemento a dos) antes de sumarlo a destino. (¡Casi nada!)

El segundo octeto es:

mm 000 *aaa*

Donde *mm* y *aaa* indican el modo de direccionamiento del punto de destino como antes hemos descrito. El 000 que aparece en el centro del octeto no es una referencia a ningún registro, sino que es parte del código de operación, ayudando a especificar que se trata de una suma.

Dependiendo de *mm* y *aaa*, pueden haber más bits indicando valores de decalage (la tabla vista más atrás da detalles de estas codificaciones). Después de la información sobre el direccionamiento viene el dato inmediato. Si $w=0$ hay 1 octeto de dato; si $w=1$ son dos los octetos del dato. Veamos qué ejemplo. El código fuente ensamblador es:

```
ADDBI   6[BX][DI],5   ; suma 5 a la posición
                      ; BX+DI+6
```

La fuente y destino son ambos números de 8 bits, por lo que $sw=00$. El destino sigue el mismo modo de direccionamiento que antes. El código máquina será:

100000 0 0   01 000 001   00000110   00000101

Nótese que el último octeto es un dato inmediato.

La forma especial para los datos inmediatos se utiliza sólo cuando el destino está en el acumulador (AX o AL). El primer octeto es:

0000010*w*

Donde *w* indica el tamaño de los datos. Si el dato es de 8 bits, $w=0$; y si es de 16, $w=1$.

Seguidamente viene el dato inmediato. Si $w=0$, el dato ocupa un octeto; si $w=1$, ocupa 2 octetos.

Como ejemplo de esta última forma, veamos el siguiente código ensamblador:

```
ADDI   AX,7   ; suma 7 a AX
```

El registro AX es de 16 bits, luego $w=1$. El código máquina será:

0000010 1   00 000 111   00000000

Nótese que de nuevo los dos últimos octetos corresponden al dato inmediato.

PROGRAMAS-EJEMPLO

En esta sección daremos un programa ejemplo corto para el 8086/8088 (véase la figura 3.20). Supondremos que trabajamos con un 8088, y utilizaremos bus de datos de 8 bits. Este programa es una subrutina que permite dibujar un punto simple (x,y) en una pantalla gráfica.

```
XMACRO-86 3.36  1

                                ; RUTINA: DOT
                                ;
                                ; ESTA RUTINA DIBUJA UN PUNTO (X,Y) SOBRE UNA PANTALLA GRAFICA
                                ; LA PANTALLA ESTA "PROYECTADA" EN LA MEMORIA, Y
                                ; TIENE 256 PIXELS HORIZONTALES POR 240 VERTICALES, CON
                                ; 4 BITS POR PIXEL PARA CODIFICAR EL COLOR.
                                ;
                                ; LA RUTINA DESTRUYE LOS REGISTROS BX, DI, Y AL.
                                ;
                                ; LA RAM PROYECCION DE LA PANTALLA OCUPA LOS PRIMEROS 32K OCTETOS
                                ; DE UN SEGMENTO DE 64K. LAS CANTIDADES XCRD (COORDENADA X)
                                ; YCRD (COORDENADA Y), MASK Y COLOR SE GUARDAN EN LA
                                ; SEGUNDA MITAD DEL SEGMENTO, MASK Y COLOR SIRVEN
                                ; PARA COLOCAR EL CODIGO CORRECTO EN EL OCTETO DE "PANTALLA RAM".
                                ; MASK Y COLOR SON EN REALIDAD PEQUEÑAS TABLAS DE BUSQUEDA
                                ; QUE SE INICIALIZAN CON UNA RUTINA SETCOLOR.
                                ;
                                ; MASK[0] CONTIENE EL NUMERO BINARIO 11110000 (USADO PARA BORRAR EL
                                ; CUARTETO INFERIOR). MASK[1] CONTIENE 00001111 (BORRA EL CUARTETO
                                ; SUPERIOR). COLOR[0] CONTIENE EL CODIGO DE COLOR EN EL CUARTETO
                                ; INFERIOR. COLOR[1] CONTIENE EL CODIGO DE COLOR EN EL CUARTETO
                                ; SUPERIOR.
                                ;
                                    EXTERNAL    XCRD,YCRD,MASK,COLOR
                                ;
0000'     BA 3E 0000*           DOT:    MOVB    BH,YCRD         ; OBTEN LA COORDENADA Y
0004'     BA 1E 0000*                   MOVB    BL,XCRD         ; OBTEN LA COORDENADA X
0008'     8B FB                         MOV     DI,BX           ; SALVAGUARDA TAMBIEN EN DI
000A'     D1 FB                         SAR     BX              ; BX TIENE LA POSICION DEL OCTETO
000C'     81 E7 0001                    ANDI    DI,1            ; DI TIENE LA POSICION DEL BIT
0010'     8A 07                         MOVB    AL,[BX]         ; OBTIENE EL OCTETO DE PANTALLA
0012'     22 85 0000*                   ANDB    AL,MASK[DI]     ; AGUJEREALO UTILIZANDO MASK DESPLAZADO
0016'     0A 85 0000*                   ORB     AL,COLOR[DI]    ; INTRODUCELO EN EL PIXEL UTILIZANDO EL
                                                                ; COLOR DESPLAZADO
001A'     88 07                         MOVB    [BX],AL         ; DEVUELVE EL OCTETO
001C'     C3                            RET                     ; VUELVE
                                        END

        XMACRO-86 3.36  S

Macros:

Símbolos:
COLOR      0018*    DOT      0000'    MASK     0014*    XCRD     0006*
YCRD       0002*

Ningún error grave
```

Figura 3.20: Programa-ejemplo.

Supongamos que la pantalla ocupa 32 K de la RAM del sistema, con 30720 octetos «proyección» de la pantalla, y que ha sido inicializada con una resolución de 256 puntos horizontales (pixels) por 240 verticales, y con 4 bits por pixel para codificar cualquiera de los 16 colores. La «pantalla RAM» se proyecta en la pantalla real de una forma muy directa. Cada cuarteto (4 bits) se asigna a un punto (pixel) de la pantalla en el mismo orden en el que se lee una página impresa: de izquierda a derecha y de arriba

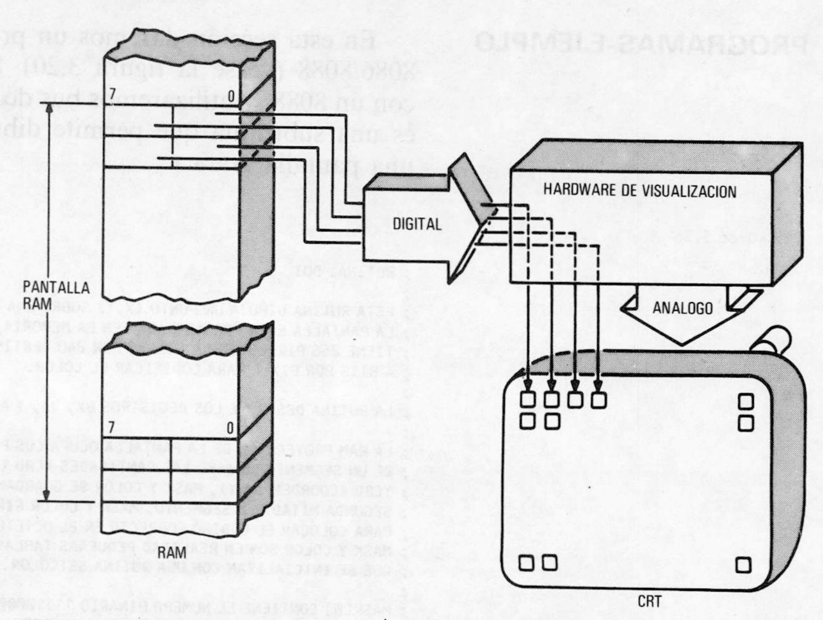


Figura 3.21: Proyección de la pantalla en la memoria del sistema.

abajo (véase la figura 3.21). El color que aparece en el pixel depende del valor almacenado en el cuarteto correspondiente. La codificación del color viene determinada por una serie de interruptores y cables hardware.

La subrutina debe calcular la dirección de cada cuarteto en la pantalla RAM e introducir el color correcto. Como la memoria es orientada a octetos, el proceso es algo complicado.

La subrutina comienza calculando la posicióñ del octeto en curso, contando a partir del comienzo de la pantalla RAM. Esto es, Y veces el número de pixels por línea (256) más X. Este cálculo se hace llevando Y a BH (el octeto superior de BX), X a BL (el octeto inferior de BX). Dividiendo este contador de cuartelo por la mitad se obtiene el desplazamiento de la dirección del octeto. Nótese que se usa desplazamiento aritmético a la derecha (SAR) para realizar la división. *No utilice nunca la instrucción DIV para estos menesteres, que es mucho más lenta.* El desplazamiento del octeto acaba en el registro BX. El segmento de datos se supone que apunta al comienzo de la pantalla, y por lo tanto se puede utilizar el modo de direccionamiento BX para acceder al octeto correcto.

El cuarteto puede estar en dos posiciones dentro del octeto. Si el contador de cuartetos es par, el cuarteto estará en la mitad inferior del octeto; si es impar, estará en la mitad superior. haciendo el AND de contador de cuartetos con 1, se obtiene la posición del cuarteto en el octeto (0=mitad inferior; 1=mitad superior). El número de posición acaba en el registro DI, que se usa como índice para dos tablas de búsqueda, MASK y COLOR. Estas tablas se inicializan vía la rutina SETCOLOR (no mostrará aquí), y carga el color para del dibujo (plotter) posterior. El

colocar el dibujo de color se hace en dos pasos. Primero se borra el cuarteto correspondiente haciendo el AND del octeto con la máscara 11110000 para el cuarteto inferior, o 00001111 para el superior: Estos valores se hallan en MASK[0] y MASK[1], respectivamente. A continuación se hace la función OR con el código de color desplazado, que se obtiene de la tabla COLOR. Por ejemplo, si el código de color es 5, COLOR[0] contiene 00000101 y COLOR[1] contiene 01010000. La figura 3.22 muestra un ejemplo.

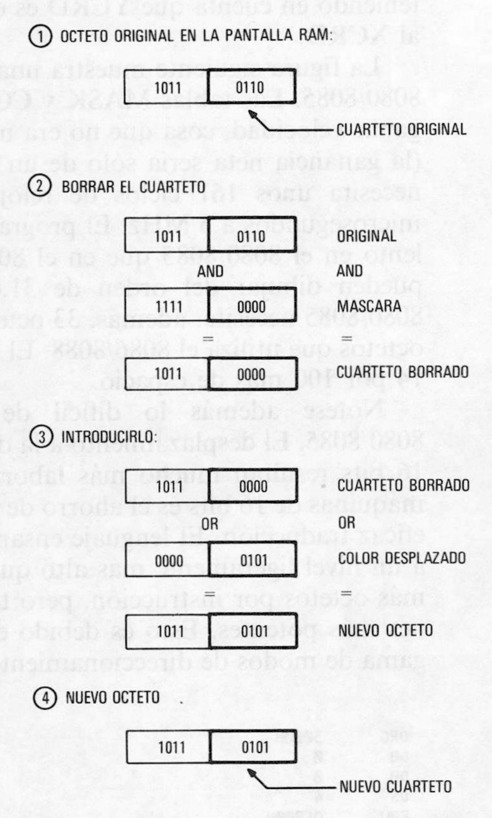


Figura 3.22: Colocación del código de color.

El programa es algo engañoso porque el valor Y no se multiplica en realidad por 256, sino que se coloca en el octeto superior de BX. Con ello se consigue que el proceso sea más rápido, cuestión importante a la hora de dibujar figuras. La subrutina emplea unos 107 ciclos de reloj. Sin embargo, puede necesitar más ciclos en caso de que la BIU no sea capaz de obtener las instrucciones a la velocidad que la EU las procesa. A 5 MHz, los 107 ciclos representan alrededor de 21,4 microsegundos. Se pueden dibujar del orden de 46.729 puntos por segundo. Por supuesto, en el dibujo total intervendrán otras rutinas software que también necesitarán tiempo para ejecutarse, de

modo que los valores citados representan tiempos mínimos. Un dibujo de 50 líneas de 100 puntos cada una puede requerir un total de unos 0,11 segundos.

El programa descrito puede mejorarse en un 10 por 100 aproximadamente cargando X e Y los dos a la vez, con la desventaja de que el programa será todavía menos claro. Para ello basta reemplazar las dos primeras instrucciones por:

```
MOV   BX,XCRD
```

teniendo en cuenta que YCRD es el octeto siguiente en memoria al XCRD.

La figura siguiente muestra una subrutina equivalente para el 8080/8085. Las tablas MASK y COLOR se han intercalado para ganar velocidad, cosa que no era necesaria hacer en el 8086/8088 (la ganancia neta sería sólo de un par de ciclos). Este programa necesita unos 161 ciclos de reloj, que representan unos 32,2 microsegundos a 5 MHz. El programa resulta un 50 por 100 más lento en el 8080/8085 que en el 8086/8088. En este caso sólo se pueden dibujar del orden de 31.056 puntos por segundo. El 8080/8085 necesita, además, 33 octetos de memoria, contra los 29 octetos que utiliza el 8086/8088. El 8080/8085 usa del orden de un 14 por 100 más de espacio.

Nótese además lo difícil de programar que resulta el 8080/8085. El desplazamiento a la derecha de 16 bits y la suma de 16 bits resultan mucho más laboriosas. La gran ventaja de las máquinas de 16 bits es el ahorro de esfuerzo de programación,y la eficaz traducción. El lenguaje ensamblador del 8086/8088 trabaja a un nivel ligeramente más alto que el del 8080/8085, generando más octetos por instrucción, pero también ofreciendo instrucciones más potentes. Esto es debido en particular a la más amplia gama de modos de direccionamiento disponibles en el 8086/8088.

```
0000                             ORG    3000H
3000 00           XCRD           DB     0
3001 00           YCRD           DB     0
3002              MASK           DS     4
3006     F800     SCREEN         EQU    OF800H
3006                                           ; RUTINA: DOT
3006                                           ;
3006                                           ; ES UNA RUTINA EQUIVALENTE PARA
3006                                           ; EL 8080/8085
3006                                           ;
3006 AF           DOT            XRA    A      ; BORRA EL ARRASTRE
3007 3A 01 30                    LDA    YCRD   ; OBTIENE LA COORDENADA Y
300A IF                          RAR           ; DIVIDE POR 2
300B 67                          MOV    H,A    ; MITAD SUPERIOR DEL DESPLAZAMIENTO DEL OCTETO
300C 3A 00 30                    LDA    XCRD   ; OBTIENE LA COORDENADA X
300F 4F                          MOV    C,A    ; LA SALVAGUARDA
3010 1F                          RAR           ; DIVIDE POR 2
3011 6F                          MOV    L,A    ; MITAD INFERIOR DEL DESPLAZAMIENTO DEL OCTETO
3012 11 00 FB                    LXI    D,SCREEN
3015 19                          DAD    D      ; SUMA A LA DIRECCION DE PANTALLA
3016 EB                          XCHG          ; DE APUNTA AL OCTETO EN LA PANTALLA RAM
3017 79                          MOV    A,C    ; OBTIENE DE NUEVO EL CONTADOR DE CUARTETOS
3018 E6 01                       ANI    1      ; SOLO EL BIT MENOS SIGNIFICATIVO
301A 87                          ADD    A      ; LO DOBLA
301B 4F                          MOV    C,A    ; PARTE INFERIOR DE BC
301C 06 00                       MVI    B,0    ; PARTE SUPERIOR DE BG
301E 21 02 30                    LXI    H,MASK ; SUMA A LA DIRECCION DE MASK
```

```
3021 09        0260        DAD    B        ; HL APUNTA A MASK
3022 1A        0270        LDAX   D        ; OBTIENE EL OCTETO DE MEMORIA
3023 A6        0280        ANA    M        ; ABRE UN AGUJERO
3024 23        0290        INX    H        ; HL APUNTA A COLOR
3025 B6        0300        ORA    M        ; INSERTA EL COLOR
3026 12        0310        STAX   D        ; LO DEVUELVE
3027 C9        0320        RET             ; VUELVE
3028           0330        END
```

Figura 3.23: Programa equivalente para el 8080/8085.

CONCLUSIONES

En este capítulo se han explorado diversas facetas de los chips microprocesadores 8086 y 8088 de Intel. Hemos visto cómo se relacionan entre ellos, y con sus predecesores de 8 bits, el 8080 y el 8085 de Intel. Los hemos comparado también con sus rivales de 16 bits, el Motorola MC68000 y el Zilog Z8000. Hemos descrito su estructura interna, incluyendo la arquitectura pipeline, sus procesadores duales en un diseño de chip simple y su juego de registros de 16 bits. Se ha comentado también la forma en la que estos chips se conectan con el mundo exterior (señales y terminales). Finalmente, hemos estudiado su juego de instrucciones más común que incluye operaciones de 8 y 16 bits, aritméticas y lógicas, y operaciones en aritmética decimal codificada en binario (con multiplicación y división con y sin signo); un juego completo de instrucciones de tratamiento de cadenas; y algunas instrucciones especiales (LOCK y ESCAPE) para el multiproceso y los procesos paralelos. Se ha visto algún programa ejemplo corto y en el capítulo 7 se verán muchos más.

En los tres capítulos siguientes discutiremos algunos chips necesarios si se quiere diseñar un microcomputador realmente potente basado en el 8086 y 8088. Comenzaremos por el Procesador de Datos Numérico 8087 en el capítulo 4, y estudiaremos, en el capítulo 5, el Procesador de E/S 8089.

Capítulo 4

El procesador de datos numérico 8087

En este capítulo estudiamos el Procesador de Datos Numérico 8087 (NDP). La figura 4.1 muestra los terminales y la estructura interna del NDP, así como la manera en que debe conectarse a la CPU 8086/8088. En el capítulo 2 ya vimos la función del NDP dentro del diagrama de bloques de una computadora basada en el 8086 de 16 bits y en el 8088 de 8 bits.

El 8087 aumenta el juego de instrucciones del 8086/8088, mejorando su capacidad de tratamiento de números. Se utiliza como procesador paralelo junto al 8086/8088, añadiendo 8 registros de coma-flotante de 80 bits a los registros del 8086/8088. Utiliza su propia cola de instrucciones para controlar el flujo de instrucciones del 8086/8088, ejecutando sólo aquellas instrucciones que le corresponden, e ignorando las destinadas a la CPU 8086/8088. Necesita la misma estructura de buses, la misma alimentación y el mismo tipo de sincronización que el 8086/8088/8089 funcionando en modo máximo, de manera que, si se desea, puede compartir la estructura de buses con la CPU 8086/8088 en modo máximo. Las instrucciones del NDP 8087 incluyen un juego completo de funciones aritméticas así como un potente núcleo de funciones exponenciales, logarítmicas y trigonométricas. Utiliza un formato interno de números en coma-flotante de 80 bits con el cual gestiona siete formatos exteriores distintos (compárese con la popular Unidad de Proceso Aritmética 9511 que tiene 32 bits como máximo para sus datos en coma-flotante). Empezaremos nuestro estudio viendo cómo se representan los números en una computadora para detallar a continuación cómo gestiona el NDP 8087 tales representaciones.

LOS NUMEROS Y SU TRATAMIENTO

Hay dos tipos de números que aparecen normalmente durante el cálculo: los números enteros y los números reales. Aunque los enteros no dejan de ser un subconjunto de los reales, la computadora trabaja de formas distintas con ambos. Los enteros son fáciles de tratar para la computadora. Los chips microprocesadores actuales de propósito general trabajan con los enteros utilizando la representación binaria de números en complemento a dos. Pueden trabajar incluso con números que excedan el tamaño de la palabra a base de fragmentar los números en unidades más pequeñas. Es lo que se llama aritmética de precisión múltiple. Los números reales, sin embargo, son más difíciles. En primer lugar, ¡la mayoría de ellos nunca pueden representarse exactamente! La

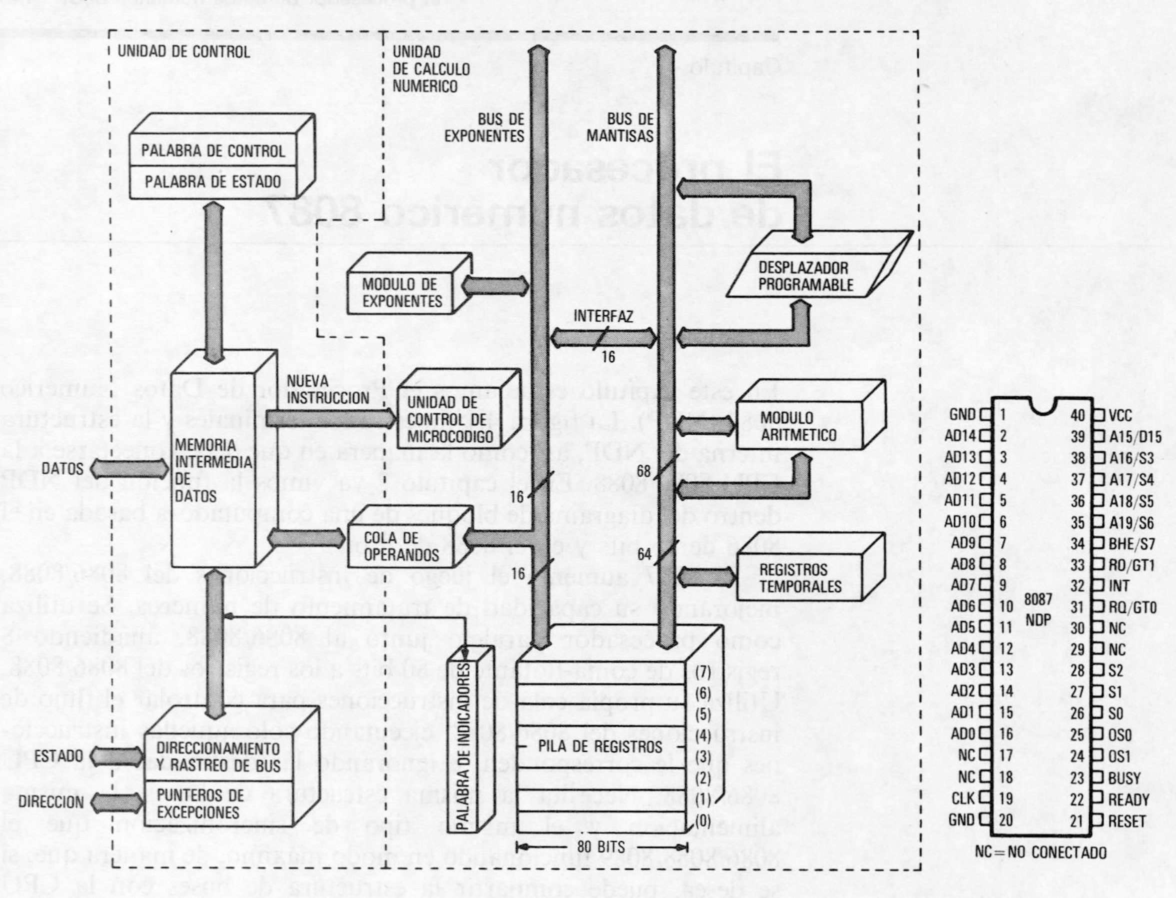


Figura 4.1a: Terminales y estructura interna del NDP 8087.

representación en coma-flotante permite una representación *aproximada* muy buena en la práctica de los números reales. La representación en coma-flotante es en el fondo una variación de la notación científica que puede verse en el visualizador de cualquier calculadora. Con este sistema, la representación de un número consta de tres partes: el signo, el exponente y la mantisa. Antes de continuar, veremos por qué es necesaria esta representación. Para los principiantes en el tema, consideremos el siguiente ejemplo de notación científica:

4,1468E2

En este ejemplo, el signo es positivo (por omisión se suponen positivos), la mantisa es 4,1468 y el exponente 2. El valor representado por este número es:

$4{,}1468 \times 10^2 = 414{,}68$

Precisión y rango

En la representación de números aparecen dos problemas fundamentales: la precisión y el rango.

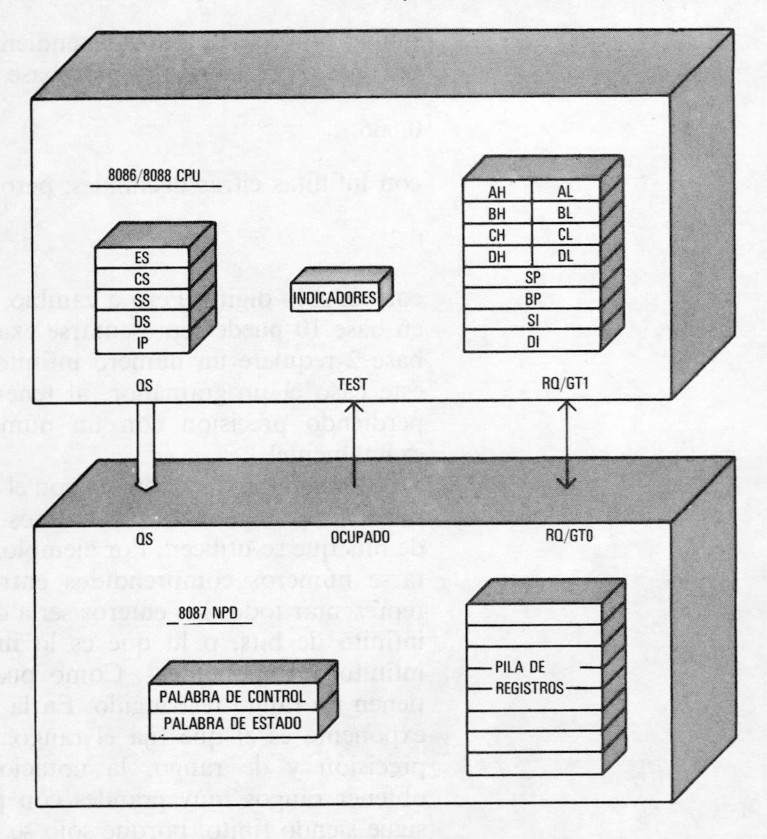


Figura 4.1b: Conexiones entre el NDP 8087 y la CPU.

Por *precisión* se entiende la exactitud de un número. En la representación en coma-flotante, la mantisa es la encargada de la precisión. La mantisa contiene los dígitos *significativos* del número independientemente de dónde esté colocada la coma decimal. Si queremos aumentar la precisión de un esquema de representación en coma-flotante, basta con añadir dígitos a la mantisa.

La precisión no es un problema en el caso de los enteros, puesto que todo entero viene representado exactamente por su complemento a dos. La representación precisa de los números reales si es un problema debido a que la mayoría de ellos (incluso la mayoría de los números reales «pequeños») tiene un número infinito de dígitos, cosa imposible de representar exactamente en una máquina con un número finito de componentes. Puesto que las computadoras permiten almacenar un número finito de dígitos, la representación de los números reales debe realizarse necesariamente por medio de *aproximaciones*.

Todavía hay más complicaciones. Números como la raíz cuadrada de 2, e y π tienen un número infinito de dígitos en cualquier base, y por tanto nunca pueden representarse exactamente. Por otro lado, hay otros números que contienen un

número finito de dígitos dependiendo de en qué base se representen. Así, por ejemplo, 2/3 en base 10 es:

0,666...

con infinitas cifras decimales; pero en base 3 es simplemente:

0,2

con un solo dígito. Pero a cambio, hay números como 1/10, que en base 10 puede representarse exactamente como 0,1, y que en base 2 requiere un número infinito de cifras. Fijémonos que en este caso el programador, al tener que utilizar la base 2, ¡está perdiendo precisión con un número que puede representarse exactamente!

El *rango* está relacionado con el tamaño de los números que se pueden representar. En los enteros, el rango depende del número de bits que se utilicen. Por ejemplo, con 16 bits pueden representarse números comprendidos entre −32.768 y +32.767. Para representar todos los enteros sería de nuevo necesario un número infinito de bits, o lo que es lo mismo, una computadora con infinitos componentes[1]. Como puede verse, incluso los enteros tienen un rango restringido. En la notación de coma-flotante, el exponente es el que fija el rango. Separando los problemas de precisión y de rango, la notación en coma-flotante permite obtener rangos muy grandes con precisión razonable. El rango sigue siendo finito, porque sólo se puede representar un número finito de dígitos del exponente, pero tales dígitos pueden representar un número bastante grande de una forma compacta.

Hemos visto que, como una computadora sólo puede asignar un número finito de bits a la mantisa y al exponente, la precisión y el rango quedan restringidos la tercera parte de la representación es el signo. En el sistema de representación coma-flotante, el signo se trata por separado de la magnitud del número, en contraposición a la representación del complemento a dos.

Implementación de la representación en coma-flotante

Hay muchas formas de implementar la representación en coma-flotante. Intel sigue el método propuesto por el instituto de normalización IEEE[2]. Para ver cómo trabaja, volvamos de nuevo a la notación científica. Por ejemplo, en notación científica decimal el número 414, 68 *suele* escribirse como:

4,1468E2

y el número −0,00345 *suele* representarse por:

−3,45E−3

[1] Incluso aunque se pudiese diseñar una computadora que fuese construyendo sus propias celdas de memoria conforme las necesitase, se necesitaría un tiempo infinito para que pudiese guardar el valor exacto de números como π o e.

[2] «A proposed Standar for Binary Floating-Point Arithmetic», *Computer* (marzo, 1981), IEEE.

En cada caso el número consta de un signo (en el primer número no aparece por ser positivo), una mantisa y un exponente. La «E» indica sencillamente «exponente» y puede leerse como «diez a la». *Normalmente,* la coma decimal se coloca a la derecha del primer dígito significativo. Cuando esto ocurre, se dice que el número está *normalizado.*

Algunos cálculos sobre estos números pueden generar resultados no normalizados. En estos casos se hace indispensable una *normalización,* que consiste en desplazar la coma decimal tantas veces a la izquierda o a la derecha como sea necesario a la vez que se incrementa o decrementa el exponente. Por ejemplo, el producto de los dos números anteriores da:

(4,1468E2) (−3,45E−3) = 14,30646E−1

Desplacemos la coma decimal a la izquierda una posición a la vez que incrementamos el exponente:

14,30646E−1 = 1,430646E0

El exponente cero es un caso particular que no admite normalización. Puede representarse por una mantisa cero. El exponente en tal caso es arbitrario, aunque se siguen ciertos convenios dependiendo de la implementación particular.

En contraste con la notación científica usual que utiliza la base 10 las computadoras utilizan la base 2 a la vez como base de la expresión exponencial, y como base para representar la mantisa y el exponente. Así por ejemplo, el número decimal 5,325, que puede escribirse en potencias de dos como:

4 + 1 + 1/4 + 1/8 = 101,011 (base 2)

se escribe en notación científica «binaria» como:

+1,01011E2

En este caso E debe leerse como «2 a la», y el incrementar o decrementar el exponente significa desplazar convenientemente la coma binaria.

Cuando se almacena un número binario en coma-flotante en la máquina, se utiliza 1 bit para el signo, varios para la mantisa y varios para el exponente. Normalmente el bit de signo precede a los demás, después vienen los bits del exponente, y por fin los de la mantisa. Puesto que el exponente puede de nuevo ser positivo o negativo. podría utilizarse la forma de complemento a dos para representarlo; sin embargo, no es lo usual. La forma más común de guardar el exponente consiste en guardarle una constante, que recibe el nombre de *«exceso».* El número que se guarda recibe el nombre de *exponente en notación de exceso.* A la inversa, dada la representación de un exponente (el exponente en notación de exceso), para recuperar su valor verdadero, basta con restarle el

exceso. Como exceso se toma un número cuyo valor es aproximadamente igual a la mitad del rango de los posibles exponentes, para poder representar más o menos el mismo número de números positivos que negativos.

Como ejemplo práctico, veamos cómo representa Intel los números reales en los registros del NDP 8087. Este formato particular recibe el nombre de *formato real temporal*. Todos los datos en el interior del NDP se almacenan de esta manera. La figura 4.2 muestra un esquema de dicho formato.

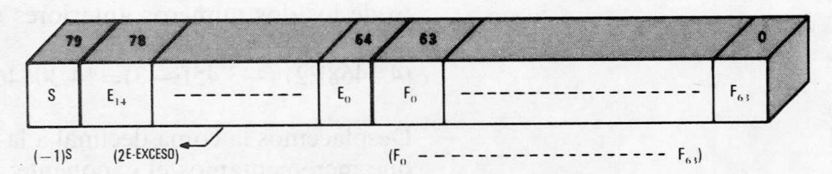


Figura 4.2: Formato interno utilizado por el NDP 8087.

El formato real temporal requiere 80 bits: uno para el signo, 15 para el exponente y 64 para la mantisa (esto es, 64 bits de precisión). El exceso es $2^{14}-1=16.383$. El número menor que se puede representar es 2^{-16382}, que es aproximadamente $3{,}36\times 10^{-4932}$. El número mayor es el 2^{16384}, que es aproximadamente $1{,}19\times 10^{4932}$.

Para hacernos una idea de la magnitud del rango de representación de números reales en coma-flotante del NDP 8087, consideremos que el radio del Universo, según las más recientes teorías, es del orden de trece billones de años-luz. Esto representa un radio de cerca de 1,23E28 centímetros, o un volumen de aproximadamente 7,74E84 centímetros cúbicos. Se necesitarían del orden de 1E122 electrones para llenar este volumen. Incluso estos números son pequeños si los comparamos por el rango permitido por el formato real temporal de Intel. Como contraste, los matemáticos suelen manejar números que exceden la capacidad de representación del NDP. Por ejemplo, todos aquellos cálculos que involucran una *función factorial* (suele representarse como n). Esta función da el número de formas posibles de ordenar n objetos (permutaciones). ¡La mayoría de las calculadoras pueden llegar a trabajar con 69! ¡Con el NDP, «sólo se puede llegar a 1754! ¡El NDP necesita menos de una décima de segundo para hacer este cálculo! (¡Menos de lo que necesita cualquier calculadora para calcular 69!)

Es importante también darnos cuenta de la precisión que ofrece el NDP 8087. Con 64 bits se tiene una precisión de $1/2^{64}$. Este número es aproximadamente la razón entre el diámetro de un átomo de hidrógeno y el de la circunferencia que describe la Luna alrededor de la Tierra. Desde luego, tal precisión no es necesaria normalmente en el resultado final; sin embargo resulta

muy importante durante los cálculos intermedios que es precisamente donde se pierde exactitud.

Mirando cuidadosamente las especificaciones del formato real temporal, se ve que en realidad se pueden representar números más grandes y más pequeños que los especificados. Intel reserva esos «números» mayores y menores que el estándar para otros propósitos. Incluso a algunos de ellos se les llama NAN, significando que «no es un número» (Not A Number). Veremos más sobre ellos en la sección dedicada a tipos de datos de NDP.

Las reglas o algoritmos de las operaciones aritméticas en coma-flotante son bastante más complejas que las de las operaciones en complemento a dos. Operaciones como la suma y la resta requieren ahora más pasos al tener que ajustar los números al mismo exponente antes de sumarlos o restarlos, y normalizar el resultado final. Intel ha desarrollado paquetes software que realizan estas operaciones en el 8080, 8085 u 8086/8088, pero son bastante lentas. Por ejemplo, el 8086 tarda del orden de 1600 microsegundos en realizar una multiplicación de números de 80 bits en coma-flotante por software. Intel tiene también una placa (Unidad Matemática Rápida iSBC310) que multiplica 10 veces más rápidamente, y un chip procesador aritmético (el 8232) que tiene una velocidad similar a la placa, con la ventaja adicional de un tamaño más reducido. El 8232, a 5 MHz, tardaría unos 32 microsegundos en multiplicar dos números de 32 ó 64 bits en coma-flotante. Y ahora, Intel ha lanzado al mercado su NDP 8087, que gana a los sistemas anteriores en cuanto a velocidad. En el caso de la multiplicación en coma flotante, la mejora es de un factor de 2; el NDP 8087 requiere un tiempo de 19 microsegundos para realizar el producto anterior.

Introduciendo tanta potencia de cálculo en tan poco espacio, Intel ha abierto el camino de interesantes aplicaciones como el control de la orientación y movimiento de robots, o potentes terminales gráficos, con las rutinas de transformación de figuras incorporadas.

EL NDP 8087 COMO PROCESADOR PARALELO

El NDP 8087 actúa también como procesador paralelo. Esto es, comparte con la CPU: 1) el mismo bus y 2) el mismo flujo de instrucciones.

Puesto que el NDP requiere el mismo tipo de sincronización, la misma alimentación y la misma estructura de bus que el 8086/8088/8089 funcionando en modo máximo, puede compartir la estructura del bus con cualquier CPU 8086/8088 en modo máximo. Más adelante veremos cómo interconectar estos chips.

Las instrucciones del NDP aparecen mezcladas con el flujo de instrucciones de la CPU. Sin embargo, como buenos hermanos, cada cual selecciona y ejecuta sólo las que le corresponden. Todas las instrucciones correspondientes al NDP comienzan con una variante de la instrucción ESCAPE del 8086/8088 (no confundirlo con el código «escape» del ASCII). La instrucción ESCAPE es una instrucción ficticia del 8086/8088, con un operando ficticio que puede especificarse con cualquiera de los 24 modos de

direccionamiento, y otro operando también ficticio que no es más que un registro. El primer octeto de la instrucción ESCAPE tiene siempre los mismos 5 bits más significativos (11011), y los tres últimos bits contienen el código para el NPD. El siguiente octeto contiene tres bits más de código (los bits del 3 al 5 indican el registro ficticio) NDP. Contiene también información sobre el direccionamiento de un posible operando. Si no hay operando, los siguientes bits del octeto pueden utilizarse como parte del código de la instrucción. La figura 4.3 muestra una instrucción típica del NDP.

Cada instrucción que lee la CPU, también la lee el NDP. Cuando la CPU lee una instrucción ESCAPE, el NDP se da por enterado que pronto tendrá que ponerse a trabajar. «Roba» los 3 bits menos significativos del octeto de la instrucción ESCAPE y deja que la CPU continúe con el octeto siguiente. El NDP roba tres bits más de este octeto. Con esto reconoce perfectamente la operación a realizar, y hace que la CPU obtenga la primera parte (octeto o palabra, dependiendo del tamaño del bus de datos) del operando con el modo de direccionamiento indicado. La CPU finalmente pasa el control del bus al NDP, que extrae los octetos de datos necesarios, hace los cálculos correspondientes, y coloca los resultados sobre el bus.

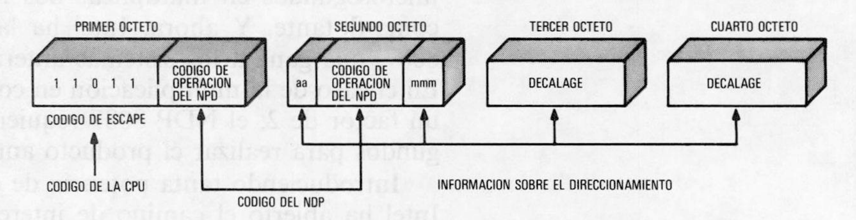


Figura 4.3: Instrucción típica del NDP.

EL NDP COMO AMPLIACION DEL 8086/8088

En términos de software el Procesador de Datos Numérico 8087 es una ampliación del 8086/8088 tanto desde el punto de vista del software, de la arquitectura, del hardware y también del comercial.

En términos de software

El NDP 8087 amplía el juego de instrucciones del 8086/8088, incluyendo operaciones en coma-flotante. Estas nuevas instrucciones implementan por hardware el paquete Intel de tratamiento de números en coma-flotante, proveyendo un método mucho mejor de ejecución de estos algoritmos. Dicho de otra manera, las rutinas de tratamiento de números en coma-flotante se pueden realizar a velocidades muy altas. Intel estima que el NDP 8087 ejecuta las operaciones de coma-flotante hasta *100 veces* más rápidamente que una CPU 8086 que los realizase por software. En la tabla 4.1 puede verse una comparación entre la velocidad del 8087 y la del paquete del software 8086. En ambos casos, los procesadores funcionaban a 5 MHz.

Instruccion	Tiempo de ejecución aproximado (microsegundos)	
	8087	8086
Suma en signo y magnitud	14	1600
Resta en signo y magnitud	18	1600
Multiplicación (precisión simple)	19	1600
Multiplicación (doble precisión)	27	2100
División	39	3200
Comparación	9	1300
Carga (doble precisión)	10	1700
Almacena (doble precisión)	21	1200
Raíz cuadrada	36	19600
Tangente	90	13000
Exponenciación	100	17100

Tabla 4.1: Velocidad del 8087 comparada con la del 8086.

En términos de arquitectura

El NDP 8087 amplía también el conjunto de registros de la CPU 8086/8088. Los registros de la CPU están en un chip (el 8086/8088), y hay ocho registros coma-flotante de 80 bits en otro chip (el 8087). La figura 4.4 da una idea del conjunto de registros ampliados desde el punto de vista del ordenador. Dichos registros guardan números en el formato *real temporal* antes estudiado.

Los ocho registros se organizan como una pila. Una pila es tal vez la mejor manera de organizar los datos para evaluar expresio-

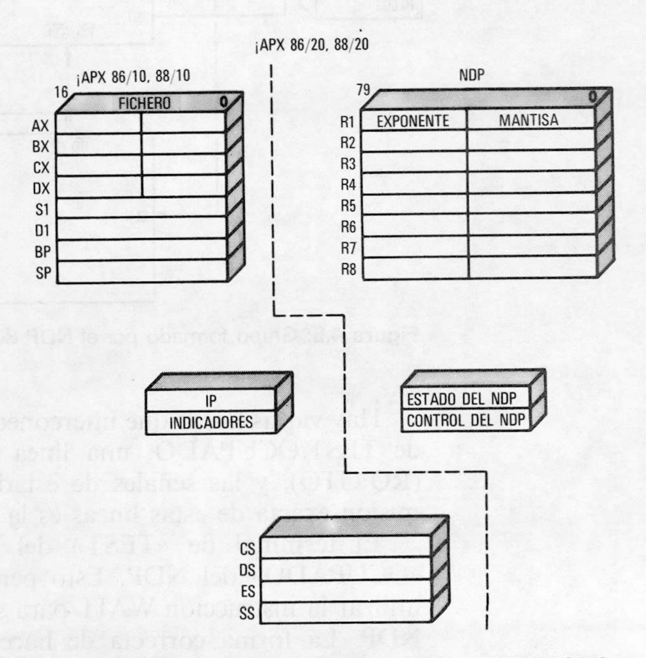


Figura 4.4: Conjunto de registros aplicado del iAPX 86/20.

nes algebraicas complejas. El NDP 8087 tiene algún registro más: un registro de estado de 16 bits, un registro de modo de 16 bits, un registro indicador de 8 bits y cuatro registros de 16 bits de salvaguarda de instrucciones y punteros a datos para la gestión de condiciones excepcionales. Esta última cuestión se explica más adelante.

En términos de hardware

El NDP es también una ampliación hardware desde el momento que no es autosuficiente. El NDP 8087 necesita tener al lado un 8086 o un 8088 que le proporcione datos y direcciones, y controle el bus que a su vez suministrará las instrucciones y operandos. La figura 4.5 muestra cómo pueden conectarse el NDP 8087 y la CPU 8086/8088. En el capítulo 2 ya vimos cómo adaptarlo a computadoras basadas en el 8086 de 16 bits o en el 8088 de 8 bits.

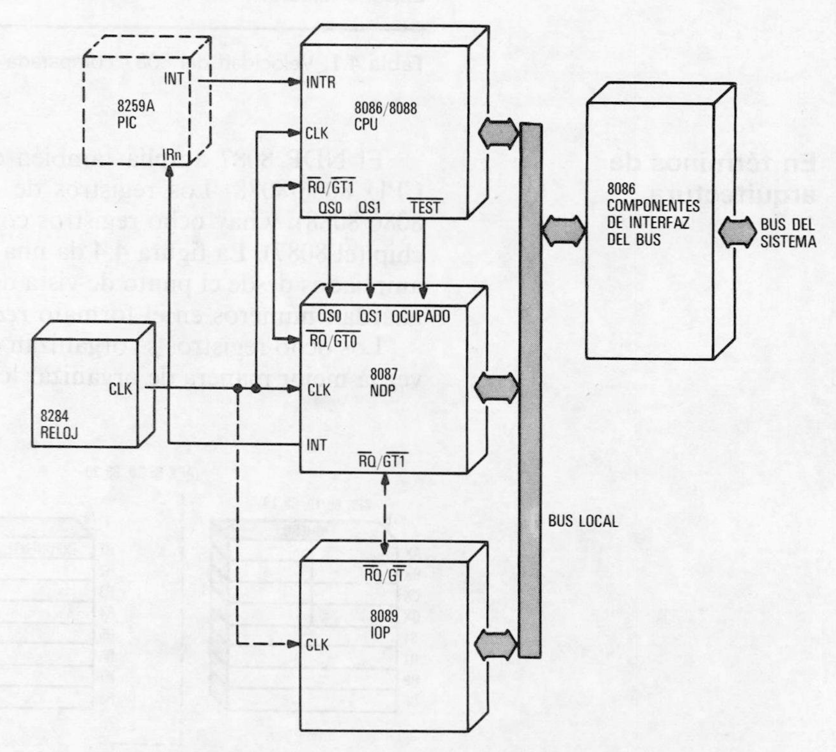


Figura 4.5: Grupo formado por el NDP 8087-IOP 8089-CPU 8086.

Hay varias líneas que interconectan el NDP y la CPU: la señal de TEST/OCUPADO, una línea de PETICION/CONCESION (RQ/GTO), y las señales de estado de la cola (QS1, QS0). La misión exacta de estas líneas es la siguiente:

El terminal de «TEST» del 8086/8088 se conecta al de «OCUPADO» del NDP. Esto permite que el 8086/8088 pueda utilizar la instrucción WAIT para sincronizar su actividad con el NDP. La forma correcta de hacerlo (al menos en el software Intel) es conseguir que el ensamblador genere automáticamente

una instrucción WAIT antes de cada instrucción del NDP, y que el programador ponga en el programa una instrucción FWAIT después de cada instrucción del NDP que cargue los datos en memoria para que puedan ser inmediatamente utilizadas por la CPU. Si en vez del NDP 8087 se utiliza el paquete de emulación software, cada instrucción numérica se convierte en una llamada a una subrutina (vía una interrupción sin WAIT), y la FWAIT se traduce por una NOP (no-operación). Si se utiliza el NDP 8087, la instrucción numérica se traduce por la correspondiente operación numérica del NDP (precedida por el WAIT) y la instrucción FWAIT se traduce por la instrucción WAIT de la CPU. Mientras el NDP 8087 está realizando una operación numérica, pone a 1 el terminal de OCUPADO (y por tanto el terminal TEST se fuerza a 1). Mientras la CPU 8086/8088 ejecuta la instrucción WAIT, suspende toda actividad hasta que el terminal TEST vuelve a su estado normal (0). Así, la secuencia de una instrucción numérica del NDP seguida de un WAIT de la CPU hace que la CPU llame al NDP y espere hasta que el NDP acabe.

La línea de PETICION/CONCESION (RQ/GTO) la utiliza el NDP para conseguir el control del bus *compartido* por el NDP y la CPU. Esta línea del NDP se conecta a una de las líneas de PETICION/CONCESION de la CPU. La línea de interconexión es bidireccional. Una señal (PETICION) del NDP a la CPU indica que la NDP quiere utilizar el bus. Para que el NDP tome control del bus, debe esperar a que la CPU le conteste (con una CONCESION). Cuando el NDP acaba de usar el bus, envía una señal a la CPU por la misma línea indicando que ha terminado. En este protocolo, la CPU juega el papel de maestro y el NDP de esclavo. El esclavo es el que pide el bus, y el maestro es el que lo concede tan pronto como puede.

Hay otra línea de PETICION/CONCESION (la RQ/GT1) en el NDP que puede utilizarse para que otros procesadores pidan el bus de la CPU/NDP. En estos casos, el dispositivo extra se comunica con el NDP a través de la línea RQ/GT1, y el NDP coloca y obtiene sus mensajes con la CPU en la primera línea de PETICION/CONCESION (RQ/GTO).

Hay dos terminales de estado de la cola, QS1 y QS0. Ambas líneas permiten al NDP sincronizar su cola de instrucciones con la de la CPU. Los dos bits se usan para codificar los cuatro estados posibles: 00=no operación, 01=primer octeto de la instrucción, 10=cola vacía, 11=octeto siguiente de la instrucción.

En términos comerciales

El NDP debe considerarse una ampliación del 8086/8088 desde el momento que no se puede comprar por separado. El 8087 sólo se consigue como parte de un conjunto de dos o tres chips, los iAPX (Intel Advanced Processor Architecture: Procesador de arquitectura avanzada Intel). El iAPX 86/20 consta de los chips de CPU 8086 y el NDP 8087; el iAPX 88/20 incluye la CPU 8088 y el NDP 8087; y el iAPX 86/21 es un conjunto de tres chips, la CPU 8086, el NDP 8087 y un 8089. La tabla 4.2 muestra la numeración de estos conjuntos de chips.

	(Incluye)			
	8086 CPU	8088 CPU	8089 IOP	8087 NDP
iAPX 86/10	Sí			
iAPX 86/11	Sí		Sí	
iAPX 86/20	Sí			Sí
iAPX 86/21	Sí		Sí	Sí
iAPX 88/10		Sí		
iAPX 88/11		Sí	Sí	
iAPX 88/20		Sí		Sí
iAPX 88/21		Sí	Sí	Sí

Tabla 4.2: Numeración de los conjuntos de chips iAPX de Intel.

TIPO DE DATOS DEL NDP 8087

El NDP puede trabajar con siete tipos de datos distintos; enteros de tres longitudes distintas, un tipo con signo, y tres tipos de representación en coma-flotante. Los números reales se representan de acuerdo con los formatos estándar IEEE de coma-flotante. Los siete chips de datos se almacenan internamente en el formato *real temporal* de 80 bits ya discutido. Todos los tipos de datos pueden guardarse con la suficiente precisión en este formato.

Sea cual sea el formato, cuando el número se obtiene de memoria se pasa del formato en el que estuviese guardado en memoria (palabra entera, entero «corto», entero «largo», BCD empaquetado, real «corto», real «largo» o real temporal) al

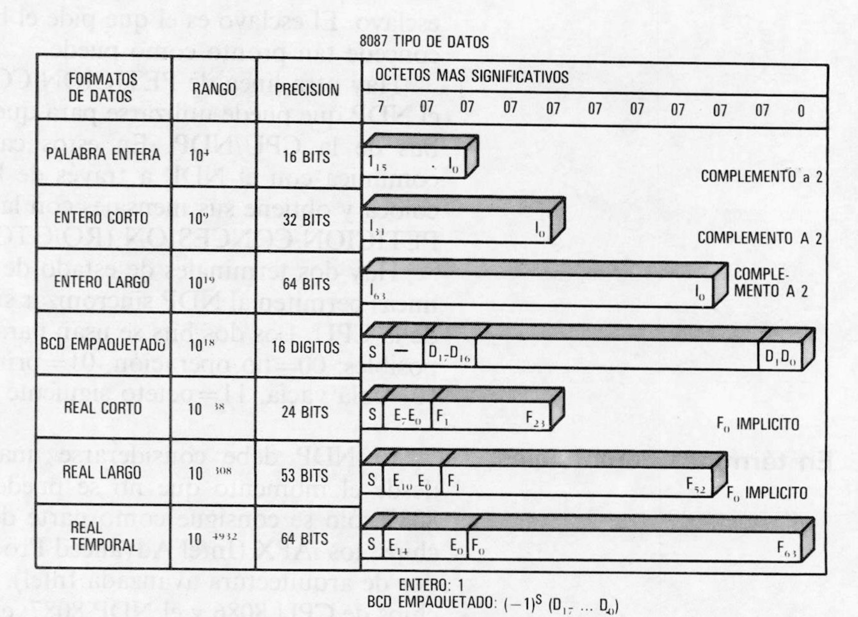


Figura 4-6: Tipos de datos del 8087.

formato real temporal utilizado interiormente. De igual manera, cada vez que un número se guarda en memoria se convierte, del formato real temporal al formato deseado.

Los formatos enteros utilizan la representación de complemento a dos. La *palabra entera* tiene 16 bits, el *entero corto* tiene 32 bits y el *entero largo,* 64 bits.

La notación BCD empaquetada tiene 18 dígitos decimales y un signo. Cada dígito se codifica en un cuarteto (4 bits). Este formato es especialmente útil en aplicaciones orientadas a los negocios, donde la precisión decimal es esencial. Permite representaciones totalmente exactas de cantidades de dólares hasta $9.999.999.999.999.999,99 (un centavo menos que 10.000 trillones).

El 8087 utiliza tres formatos en coma-flotante distintos: real corto, real largo y real temporal.

El formato real corto necesita 32 bits: uno para el signo, ocho para exponente y veintitrés para la mantisa. El exceso es $2^7-1=127$. El rango de los exponentes en notación de exceso (tal como se guardan en memoria) va de 1 a 254. El número positivo más pequeño que se puede representar es el $2^{-126}\simeq 1{,}175\times 10^{-38}$ y el mayor es 2^{128}, aproximadamente igual a $3{,}40\times 10^{38}$. Hay realmente 24 dígitos binarios de precisión, porque el primer bit, que es siempre 1, no se almacena. Sólo se guardan los veintitrés dígitos binarios a la derecha de la coma. Con una mantisa en 24 bits se consigue una precisión de site dígitos decimales y medio. El cero es un caso muy especial al no poderse normalizar, y una mantisa cero nunca es posible debido al primer dígito no almacenado que se supone 1. El cero se representa por *aproximación* a un número muy pequeño, menor que el rango permitido. En este caso se utiliza un exponente de -127 (en notación con exceso, exponente 0), y la mantisa se toma igual a 1,0, que viene representada por todo ceros (recordemos que el primer bit no se guarda). El signo puede ser positivo o negativo.

El formato real largo requiere 64 bits: una para el signo, once para el exponente y cincuenta y dos para la mantisa. El exceso es $2^{10}-1=1.023$. El exponente en notación de exceso toma valores entre 1 y 2.046. El menor número positivo representable es $2^{-1022}\simeq 2{,}23\times 10^{-308}$, y el mayor es $2^{1024}\simeq 1{,}80\times 10^{308}$. Hay realmente 53 dígitos binarios de precisión debido al dígito más significativo que no se almacena (siempre 1). Sólo se guardan 52 dígitos a la derecha de la coma. Con todos estos bits, se consigue una precisión de casi quince dígitos decimales. El cero, en este caso, se representa por el menor exponente posible (-1023, representado por el exponente 0 en notación de exceso), la mantisa 1,0 (representado como todo ceros), y el signo puede ser + o −.

Hemos discutido ya el formato real temporal, pero para completar el tema lo volvemos a tratar aquí. Decíamos que necesitaba 80 bits: uno para el signo, 15 para el exponente y 64 para la mantisa. El exceso es de $2^{14}-1=16383$. El rango del exponente en notación de exceso es de 1 a 16382. El menor

número positivo representable es el $2^{-16382} \simeq 3{,}36 \times 10^{-4932}$, y el mayor $2^{16384} \simeq 1{,}19 \times 10^{4932}$. El formato real temporal se diferencia de los otros en que se almacenan todos los bits de la mantisa. Los 64 bits de precisión equivalen a más de 18 dígitos decimales. El cero se representa por una mantisa 0 (ahora sí puede valer exactamente 0), y el mínimo exponente (−16383, equivalente a 0 en notación de exceso). Como en los otros formatos reales, el 0 puede ser positivo o negativo.

Existen ciertas «configuraciones» asociadas a cada tipo de datos, que no corresponden a números reales normalizados. Las configuraciones que corresponden a la representación en coma flotante normalizada son las llamadas *normales*. Entre las no normales están las paranormales, anormales, infinitas, NAN (No es un número) e indefinidas. Aquí nos limitaremos a describirlas muy brevemente. Para una discusión más detallada ver *Numerics Supplement* en *The 8086 Family User's Manual*. Estas configuraciones especiales, o bien tienen exponentes de tamaño mayor al máximo permitido para las representaciones estándar, o bien tienen exponentes menores que el mínimo (graduados como 0 en notación en exceso). Los paranormales y anormales responden a situaciones de desbordamiento de la capacidad mínima que permitan al NDP pasar por varias etapas de alerta cuando comienza a calcular números demasiado pequeños. Los infinitos se tratan de maneras distintas dependiendo del estado de un bit de la palabra de control. Los NAN son útiles para tareas de depuración. Hay suficientes configuraciones de este estilo para prever todos los comportamientos erróneos. La indefinida es útil como respuesta a ciertas condiciones, como cero dividido por cero. Estos «números» extras representan un buen arsenal para el NDP a la hora de gestionar situaciones irreales como la división por cero, o los desbordamientos de capacidad, sea máxima o mínima.

Resulta interesante comparar el chip NDP 8087 con una gran computadora como puede ser el Cyber de Control Data Corporation. El procesador central del Cyber tiene tres tipos de registros diferentes (véase la figura 4.7): los de dirección (A), índice (B) y datos (X). Los registros de dirección son de 18 bits (compárese con la mayor capacidad de direccionamiento —20 bits— de los chips del iAPX); los registros índice tienen también 18 bits y los de datos 60 (los del NDP pueden llegar a 80). Los registros B se utilizan para guardar números pequeños como índices de bucles. Los registros X guardan números enteros o de coma flotante. El formato entero recibe el nombre de *complemento a 1*. Las reglas de trabajo son similares a las de la aritmética de complemento a dos con ligeras variaciones. En particular, los números negativos se obtienen simplemente complementando (a 1) el positivo, y la suma se realiza sumando los números y añadiendo al resultado el acarreo generado en

los dígitos más significativos. La notación de coma-flotante utiliza los 60 bits, reservando un bit para el signo, 11 para el exponente y 48 para la mantisa. Es similar al formato real largo del NDP 8087. Aunque el NDP de Intel tiene la ventaja de ser más pequeño y tener una capacidad de direccionamiento de datos mayor, a las grandes computadoras todavía les queda la ventaja de la velocidad. Por ejemplo, una multiplicación en coma-flotante necesita del orden de 19,4 microsegundos en el NDP, ¡y tarda sólo un microsegundo en el Cyber! En contrapartida un sistema Cyber completo puede ocupar más de 260 pies cuadrados de superficie (700 pies cuadrados si se toma en cuenta el espacio necesario para la alimentación, aire acondicionado y partes relacionadas), pesa del orden de 23.000 libras, consume alrededor de 60.000 watios y cuesta sobre 9.500 dólares por mes más unos 8.600 dólares adicionales de mantenimiento. En comparación, un sistema completo basado en el NDP 8087 y la CPU 8086/8088 puede pesar unas 120 libras, ocupar algo así como 15 pies cuadrados de superficie (cabe encima de una mesa), consume unos 200 watios y cuesta del orden de 5.000 dólares.

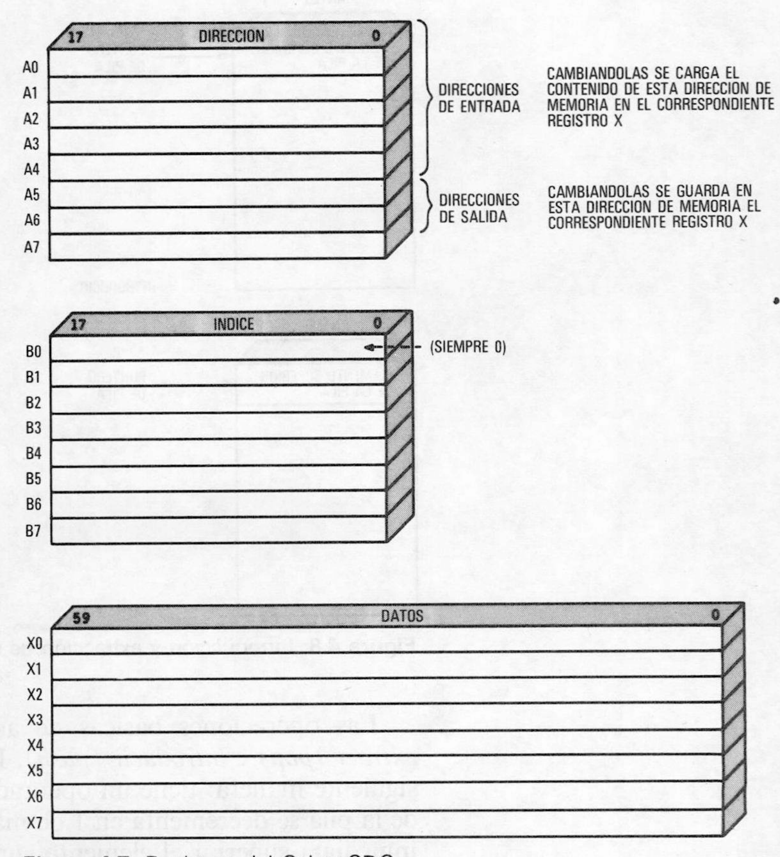


Figura 4.7: Registros del Cyber CDC.

UTILIZACION DE LA PILA DEL NDP

Una pila es como una caja en una mesa. Es una forma conveniente de guardar campos temporalmente mientras se trabaja con ellos. En una pila, el último campo que *entra* es el primero que *sale*. Las calculadoras como las Hewlett Packard que usan la notación polaca inversa utilizan una pila. Resulta rápido acostumbrarse a trabajar con un sistema como éste, y resulta fácil aprender cómo evaluar expresiones algebraicas introduciendo resultados intermedios en una pila hasta que vuelvan a ser necesarios.

Una *pila* es una *estructura de datos*. Es decir, hay reglas que asignan posiciones de memoria a los datos, y reglas para acceder a ellos. En una pila, los datos se guardan en posiciones consecutivas de memoria (en este caso, en registros consecutivos del NDP). Otra posición de memoria, llamada el puntero de pila, contiene la dirección de una de esas posiciones de los datos. A dicha posición se le da el nombre de *elemento superior de la pila*. El elemento superior de la pila, y las posiciones siguientes se consideran parte de la pila, mientras que las posiciones por encima de éstas no pertenecen a la pila. Las posiciones se suelen numerar en orden creciente desde la parte superior de la pila hacia abajo.

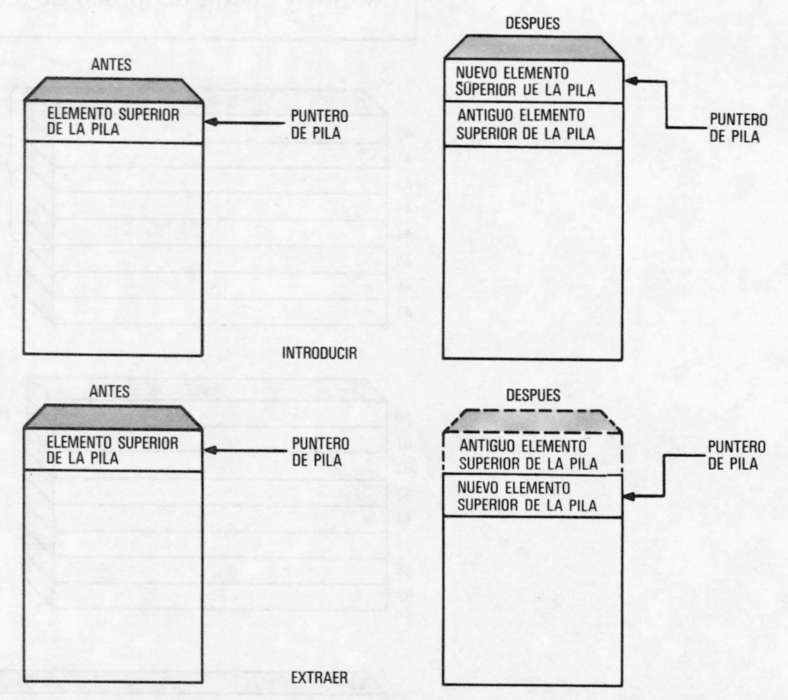


Figura 4.8: Introducción y extracción de elementos en una pila.

Las operaciones básicas de acceso a los datos son las de *extraer (pop)* e *introducir (push)*. La introducción funciona de la siguiente manera: tiene un operando que es la fuente. El puntero de la pila se decrementa en 1 de manera que apunta a la posición inmediata superior el elemento superior de la pila, y a continua-

ción el dato se transfiere a este nuevo elemento superior. El extraer funciona de la siguiente manera: En primer lugar se transfiere el dato correspondiente al elemento superior de la pila a su destino, y seguidamente el puntero de la pila se incrementa en 1, de manera que apunta ahora al nuevo elemento superior de la pila (que está en una posición por encima del anterior). El antiguo elemento superior de la pila no vuelve a formar parte de ésta, por lo que la nueva pila posee un elemento menos. Véase la figura 4.8 en la que se esquematizan ambas operaciones.

Igual que resulta peligroso apilar demasiados objetos, hay que ir con cuidado de no introducir demasiados elementos en una pila debido a los límites de memoria. La pila del NDP tiene capacidad para ocho elementos. Normalmente es suficiente. De hecho, las calculadoras Hewlett Packard tienen pilas de sólo cuatro elementos.

En el NDP y los RPN, las pilas se utilizan para guardar resultados intermedios durante la evaluación de expresiones algebraicas. También se utilizan en otros contextos; por ejemplo, la CPU utiliza una pila para salvaguardar la dirección de vuelta en las llamadas a subrutinas.

En el NDP, la pila está numerada respecto a su elemento superior. El elemento superior se denota por ST (0), o simplemente ST, y las posiciones inferiores se denotan respectivamente por ST(1), ST(2), ... ST(7). (Véase la figura 4.9.)

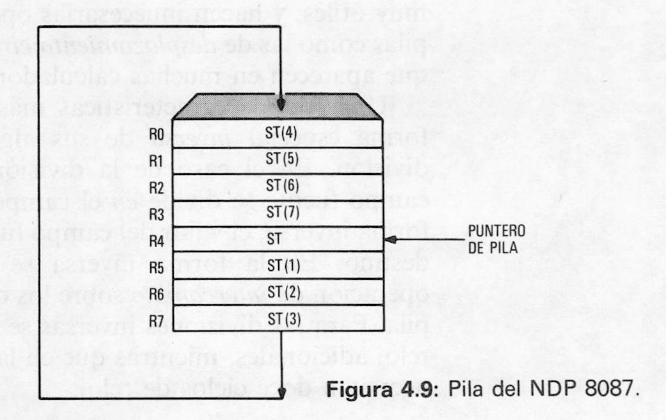


Figura 4.9: Pila del NDP 8087.

La instrucción FLD (LoaD) introduce un elemento en la pila; y la instrucción FSTP (Store and Pop) lo extrae. (Nota: los nemotécnicos de las instrucciones del NDP comienzan por F indicando la notación coma-flotante.) Hay muchas otras instrucciones similares. De hecho, la idea de pila del NDP es una *extensión* de la idea de pila mucho más estricta que acabamos de ver. Así, por ejemplo, el NDP tiene instrucciones como la FST que transfiere a una posición dada el contenido del elemento superior de la pila, pero sin incrementar el valor del puntero de pila (realmente no extrae); la FXCH (de intercambio); FINCSTP (INCrement STack Pointer: Incrementar el puntero de pila); o FDECSTP (DECrement STack Pointer: Disminuir el puntero de

pila). La instrucción FXCH intercambia el contenido del elemento superior de la pila con el de cualquier otro elemento de la pila. El resto de las operaciones citadas permiten ajustar el valor del puntero de pila sin transferir datos.

Las operaciones tales como suma, resta, multiplicación y división, que necesitan dos *argumentos de entrada* (dos operandos) tienen distintas formas. Si se especifican sin operandos, entonces la operación se realiza entre los elementos ST y ST(1) de la pila. Tras la operación, se extraen dos elementos de la pila, y el resultado se coloca en el elemento superior de ésta. Dicho con otras palabras, se sustituyen los dos operandos por el resultado de la operación. Es una operación bastante clásica en pilas; es por ejemplo, lo que sucede cuando se realiza una operación de dos operandos en notación polaca inversa.

Si sólo hay un operando, la operación se realiza entre éste y el elemento superior de la pila. El resultado reemplaza a este último.

Si hay dos operandos, debe tratarse de dos registros del NDP. Uno de ellos debe ser obligatoriamente el elemento superior de la pila, y el otro puede ser cualquier registro del NDP. Pueden darse tres casos distintos: 1) la fuente ST y el destino es ST(i), 2) la fuente es ST(i) y el destino ST, y 3) la fuente es ST, el destino ST(i), y se extrae un elemento de la pila. En los dos primeros casos la fuente no se ve afectada, pero en el tercero, la fuente (ST) se pierde al extraer un elemento de la pila. Las tres opciones son muy útiles, y hacen innecesarias operaciones de manipulación de pilas como las de *desplazamiento circular ascendente y descendente* que aparecen en muchas calculadoras.

Unas de las características más interesantes del NDP es su forma especial *inversa* de sus algoritmos de multiplicación y división. En el caso de la división, normalmente el valor del campo fuente se divide *en* el campo de destino, mientras que la forma inversa, el valor del campo fuente se divide *por* el campo de destino. En la forma inversa se elimina la necesidad de la operación de *intercambio* sobre los dos elementos superiores de la pila. Para las divisiones inversas se necesitan uno o dos ciclos de reloj adicionales, mientras que en la operación de intercambio se necesitan doce ciclos de reloj.

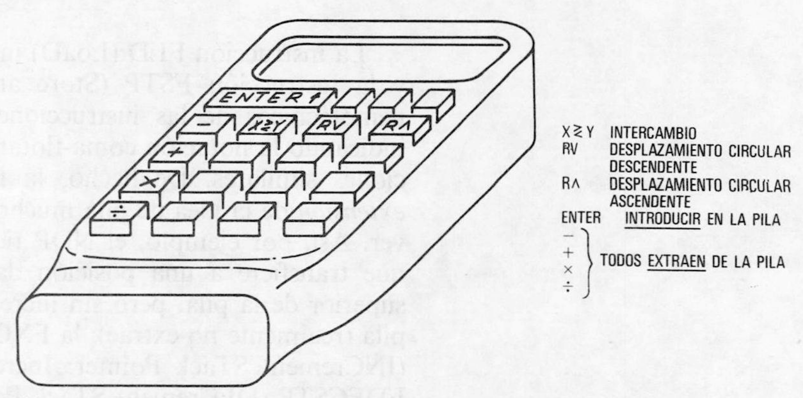


Figura 4.10: Operaciones de pila en el RPN.

JUEGO DE INSTRUCCIONES DEL NDP 8087

El NDP 8087 tiene 48 tipos de instrucciones distintas, incluyendo transferencia y conversión de datos, comparaciones, las cuatro operaciones aritméticas básicas, exponenciales y logaritmos, constantes especiales y de control. La tabla 4.3 resume el juego de instrucciones del NDP 8087.

Tabla 4.3

Nemotécnico		**Descripción**
FLD	fuente	Cargar de memoria a ST (coma flotante)
FST	destino	Guardar ST en memoria (coma flotante)
FSTP	destino	Guardar ST en memoria y extraer un elemento de la pila (coma flotante)
FXCH	fuente	Intercambiar ST(i) y ST (coma flotante)
FCOM		Comparar ST(1) y ST (coma flotante)
FCOM	fuente	Comparar la posición de memoria especificada con ST o ST(i) con ST (coma flotante)
FCOMP		Comparar ST(1) con ST y extraer un elemento de la pila (coma flotante)
FCOMP	fuente	Comparar la posición de memoria especificada con ST o ST(i) con ST, y extraer un elemento de la pila (coma flotante)
FCOMPP		Comparar ST(1) con ST y extraer dos elementos de la pila (coma flotante)
FTST		Verificar ST (coma flotante)
FXAM		Examinar ST (coma flotante)
FADD		Sumar ST(1) a ST (coma flotante)
FADD	fuente	Sumar el valor de la posición de memoria especificada a ST (coma flotante)
FADD	destino, fuente	Sumar ST(i) a ST o ST a ST(i) (coma flotante)
FSUB		Restar ST(1) a ST (coma flotante)
FSUB	fuente	Restar el valor de la posición de memoria especificada a ST (coma flotante)
FSUB	destino, fuente	Restar ST(i) a ST o ST a ST(i) (coma flotante)
FSUBR		Resta inversa de ST(1) a ST (coma flotante)

Tabla 4.3

FSUBR	fuente	Resta inversa del valor de la posición de memoria especificada a ST (coma flotante)
FSUBR	destino, fuente	Resta inversa de ST(i) a ST o de ST a ST(i) (coma flotante)
FMUL		Multiplicar ST(1) por ST (coma flotante)
FMUL	fuente	Multiplicar el valor de la posición de memoria especificado por ST (coma flotante)
FMUL	destino, fuente	Multiplicar ST(i) por ST y poner el resultado en ST(i) o en ST según destino (coma flotante)
FDIV		Dividir ST(1) por ST (coma flotante)
FDIV	fuente	Dividir el valor de la posición de la memoria especificada por ST (coma flotante)
FDIV	destino, fuente	Dividir ST(i) por ST o ST por ST(i) (coma flotante)
FDIVR		División inversa de ST(1) por ST (coma flotante)
FDIVR	fuente	División inversa entre el valor de la porción de memoria especificada y ST (coma flotante)
FDIVR	destino, fuente	División inversa de ST(i) por ST o ST por ST(i)
FSQRT		Raíz cuadrada de ST (coma flotante)
FSCALE		Escalado de ST por ST(1) (coma flotante)
FPREM		Resto parcial (coma flotante)
FRNDINT		Redondeo de ST a entero (coma flotante)
FXTRACT		Extraer componentes (coma flotante)
FABS		Valor absoluto de ST (coma flotante)
FCHS		Cambio de signo de ST (coma flotante)
FPTAN		Tangente parcial de ST (coma flotante)
FPATAN		Arco tangente parcial de ST (coma flotante)
F2XM1		Potencia de 2 menos 1 (coma flotante)
FYL2X		Y por log base 2 de X (coma flotante)
FYL2XP1		Y por log base 2 de X+1 (coma flotante)

FLDZ	Cargar 0.0 en ST (coma flotante)
FLD1	Cargar 1.0 en ST (coma flotante)
FLDPI	Cargar PI en ST (coma flotante)
FLDL2T	Cargar log base 2 de 10 en ST (coma flotante)
FLDL2E	Cargar log base 2 de e en ST (coma flotante)
FLDLG2	Cargar log base 10 de 2 (coma flotante)
FLDLN2	Cargar log base e de 2 (coma flotante)
FINIT	Inicializar el NDP
FENI	Activar interrupciones (coma flotante)
FDISI	Desactivar interrupciones (coma flotante)
FLDCW	Cargar palabra de control (coma flotante)
FSTCW	Guardar la palabra de control (coma flotante)
FSTSW	Guardar la palabra de estado (coma flotante)
FCLEX	Borrar excepciones (coma flotante)
FSTENV	Guardar el medio (coma flotante)
FLDENV	Cargar el medio (coma flotante)
FSAVE	Salvaguardar el estado (coma flotante)
FRSTOR	Reestablecer el estado (coma flotante)
FINCSTP	Incrementar el puntero de pila (coma flotante)
FDECSTIP	Disminuir el puntero de pila (coma flotante)
FFREE	Liberar ST(i)
FNOP	No operación (coma flotante)
FWAIT	Orden de espera de la CPU hasta que el NDP haya acabado

Tabla 4.3: Resumen de las instrucciones del NDP 8087.

Notemos que en todos los casos las instrucciones comienzan con una instrucción ESCAPE de la CPU tal como se apuntó anteriormente. La instrucción ESCAPE, recordemos (no se confunda con el código ESCAPE del ASCII) es una instrucción ficticia para el 8086/8088, con un operando ficticio que se puede especificar en cualquiera de los 24 modos de direccionamiento de la CPU. Veamos los distintos campos de estas instrucciones. En este contexto, un *campo* es un grupo de bits consecutivos, agrupados codificando una cierta información. Véase la figura siguiente.

	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
(1)	OCTETO DE DIRECCION MAS BAJA								OCTETO DE DIRECCION MAS ALTA								0, 1 O 2 OCTETOS
(2)	1	1	0	1	1	OP-A		1	aa		1	OP-B		mmm			DECALAGE
(3)	1	1	0	1	1	FORMATO		OP-A	aa		OP-B			mmm			DECALAGE
(4)	1	1	0	1	1	R	P	OP-A	1	1	OP-B			rrr			
	1	1	0	1	1	0	0	1	1	1	1	OP					
(5)	1	1	0	1	1	0	1	1	1	1	1	OP					
	ESCAPE					CODIGO NDP											

CAMPOS

Figura 4.11: Formatos de codificación de las instrucciones de 8087.

El primer octeto de todas las instrucciones del NDP tiene siempre los mismos cinco bits más significativos (código ESCAPE 11011), y los tres bits más bajos contienen el código. Este primer octeto tiene por tanto dos campos, el campo ESCAPE y el campo de código NDP. Hay en realidad cinco formatos distintos para las instrucciones del NDP, dependiendo entre otras cosas de cuantos operandos explícitos haya. Las instrucciones de un solo operando tienen su código de operación dividido en dos campos: el primero contenido en los tres bits más bajos del primer octeto, y el segundo contenido en los bits 3 a 5 del segundo octeto. Este segundo campo se reserva normalmente para designar el segundo operando (que debe ser un registro) en las instrucciones de doble operando de la CPU. Todas las instrucciones del NDP tienen como máximo un operando (externo) con lo que el campo de segundo operando queda libre para utilizarse como parte del código de operación (lo mismo se hace con las instrucciones de la CPU de un solo operando).

Tanto las instrucciones del NDP como de la CPU de no referencia a memoria tienen un 11 en los bits más significativos del segundo octeto de la instrucción. En las instrucciones de la CPU, este modelo indica el modo de direccionamiento de registro; es decir, que el operando está en un registro. Para las instrucciones del NDP puede significar o bien que el o los operandos están en registros del NDP, o que no hay operandos. La figura anterior muestra cómo están organizados esos cinco posibles formatos.

Hemos discutido ya la suma, resta, multiplicación y división en el apartado dedicado a la pila. De particular interés resulta las formas inversas para la resta y división, por el ahorro en tiempo y en pasos de programación que producen.

Hay también un conjunto de instrucciones dedicadas a cargar en la pila ciertas constantes, como el 0, el 1, el número pi, y varias constantes logarítmicas. En vez de guardar estos valores en memoria y tener que ir a buscarlos cada vez que se necesiten, el NDP ahorra un tiempo de proceso importantísimo ¡guardándolos en silicio!

El NDP puede trabajar también en aritmética de complemento a dos de 16 bits, aunque no resulta recomendable al ser bastante más lento que la CPU. Una suma registro-a-registro de

16 bits, por ejemplo, consume tres ciclos de reloj de la CPU 8086/8088, y ¡185 ciclos del NDP! Por supuesto, la razón de tan bajo rendimiento estriba en que el NDP utiliza el formato real temporal de 80 bits para procesar estos números de 16 bits (y hace las conversiones entre enteros en este mismo formato). La ventaja de que el NDP pueda trabajar con números pequeños reside en los cálculos mixtos, es decir, cálculos que operan con números enteros y reales a la vez. Puesto que en el fondo el NDP guarda todos sus datos en el mismo formato interno, cualquier combinación de los siete tipos de datos distintos puede aparecer en un cálculo.

Muchas funciones, como la raíz cuadrada FSQRT, valor absoluto FABS, y las funciones trigonométricas y logarítmicas, no pueden tener sus operandos en memoria, y trabajan sólo con el elemento superior de la pila. Con esto se crea el típico «cuello de botella» de las máquinas de estructura basada en registros; sin embargo, no resulta tan grave como parece gracias a la instrucción FXCH, que permite intercambiar cualquier registro del NDP con el primer elemento de la pila. Por ejemplo, para calcular el valor absoluto de cualquier registro, se intercambia éste con el elemento superior de la pila con la instrucción FXCH, se halla el valor absoluto, y se vuelven a intercambiar los valores. No es demasiado lento, ni demasiado difícil. De esta manera, la pila puede verse como un área de memoria de trabajo general, algo así como una RAM temporal dentro del NDP.

Otras operaciones del NDP, como algunas trigonométricas, devuelven resultados *parciales,* en el sentido de que estas funciones constituyen un *núcleo* a partir del cual se pueden obtener funciones más complicadas. Por ejemplo, la función FPTAN (tangente parcial) toma como argumento el elemento superior de la pila (el ángulo) y genera dos resultados: X e Y. Y reemplaza al primer elemento de la pila y a continuación se introduce X de manera que al acabar la operación aparecen X e Y como primer y segundo elementos de la pila, respectivamente. Las seis funciones trigonométricas estándar: seno, coseno, tangente, secante, cosante y cotangente se obtiene fácilmente a partir de X e Y. Así la tangente es Y/X, y requiere sólo una división más; el seno es Y/SQR (X∗X+Y∗Y), y el coseno es X/SQR (X∗X+Y∗Y) (utilizando el teorema de Pitágoras). Las otras tres funciones son las recíprocas de éstas y se pueden obtener utilizando la división inversa. Hay una complicación adicional: el argumento de la función FPTAN debe estar comprendido entre 0 y pi/4 (de 0 a 90 grados), de forma que el ángulo original debe reducirse al primer cuadrante antes de aplicarle la función FPTAN. El segundo ejemplo muestra cómo programar estas funciones. Evidentemente, si utilizamos una tabla de búsqueda en la que estuviesen almacenados los valores de estas funciones, el proceso sería más rápido; pero a cambio ocuparíamos una cantidad de memoria tremenda.

Ciertas instrucciones del NDP se han optimizado para que se puedan utilizar fácilmente en el cálculo de otras funciones. Es el

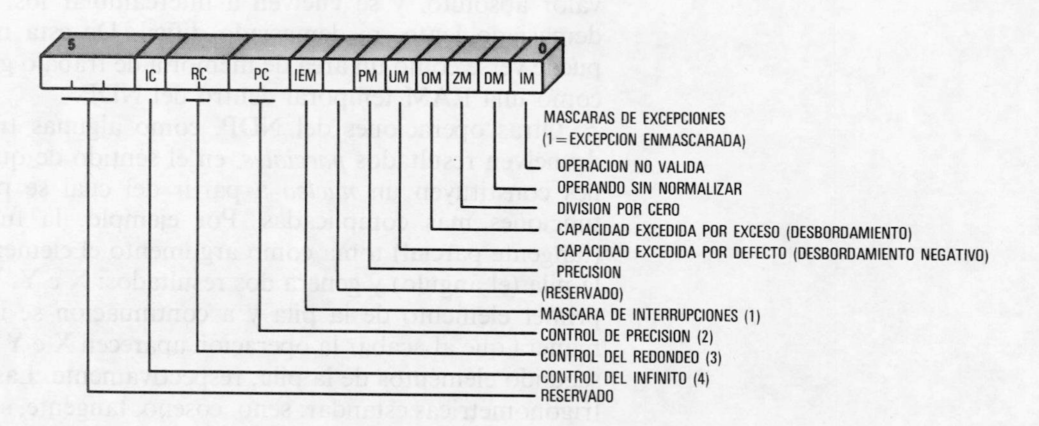
caso, por ejemplo, de la operación logaritmo, que calcula en realidad la función Y∗LOG 2 X, donde X e Y son el primer y segundo elemento de la pila respectivamente. El valor Y∗LOG 2 X puede ser útil para calcular cualquier potencia de X (X∗∗Y en notación FORTRAN), como potencia de 2, haciendo:

X∗∗Y=2∗∗(Y∗LOG 2 X)

Otras instrucciones se han mejorado para aumentar la precisión, como la FYL2XP1 (Y∗LOG 2 (X+1)), que proporciona valores precisos en el cálculo de logaritmos de números cercanos al 1.

PALABRAS DE CONTROL Y DE ESTADO

La *palabra de control* es un registro de 16 bits del NDP (véase la figura 4.12) que permite especificar cuestiones como el grado de precisión, tipo de redondeo, o incluso, ¡cómo tratar el infinito! La palabra de control especifica también cómo *manejar las condiciones excepcionales* (recuperación de errores). Las condiciones excepcionales se verán en la subsección siguiente. La palabra de control se carga en el NDP desde memoria mediante una cierta



(1) Máscara de interrupciones:
0 = Interrupción activa
1 = Interrupción inactiva (enmascarada)

(2) Control de precisión:
00 = 24 bits
11 = (reservado)
10 = 53 bits
11 = 64 bits

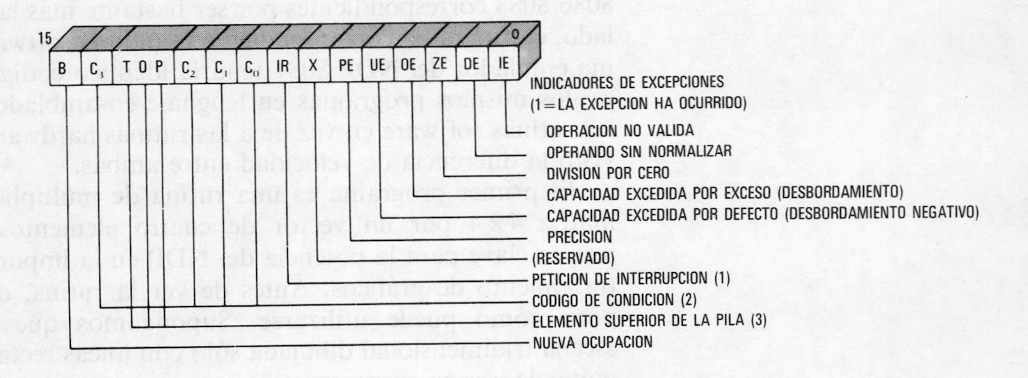
(3) Control de redondeo:
00 = Redondeo al más cercano o par
01 = Redondeo por defecto
10 = Redondeo por exceso
11 = Trunca hacia cero

(4) Control del infinito:
0 = Proyectivo
1 = Afín

Figura 4.12: Palabra de control del NDP 8087.

instrucción de control del NDP. La CPU puede definir las condiciones de la palabra de control modificando la palabra de control cuando está en memoria, y haciendo que el NDP la lea a continuación.

La *palabra de estado* es otro registro de 16 bits del NDP (véase la figura 4.13) que da información sobre el tipo de problemas que hayan surgido en el último error; el cuadrante del ángulo en funciones trigonométricas, el estado de los indicadores resultantes de una instrucción de comparación, el valor del puntero de pila, y el estado de la señal de ocupación. La palabra de estado se utiliza sobre todo en la determinación de bifurcaciones condicionales después de una comparación. La palabra de estado se carga en memoria mediante la instrucción FSTSW, y desde allí puede consultarla la CPU.



(1) IR se pone a 1 si cualquier bit de una excepción no enmascarada se pone a 1, y a 0 en caso contrario.
(2) Véase la tabla 3 para interpretar el código de condición.
(3) Valores superiores:
000 = El registro 0 es el elemento superior de pila.
001 = El registro 1 es el elemento superior de pila.
•
•
•
111 = El registro 7 es el elemento superior de pila.

Figura 4.13: Palabra de estado del NDP 8087.

GESTION DE CONDICIONES EXCEPCIONALES

El NDP 8087 interrumpe al 8086/8088 cuando encuentra un error. Es parte de la gestión de condiciones excepcionales, nombre que engloba a los procedimientos de tratamiento de los posibles errores. En el NDP 8087 hay seis excepciones: operación no válida, operación sin normalizar, división por cero, capacidad excedida por exceso (desbordamiento), por defecto y resultado inexacto. Si el NDP produce alguna de estas exepciones y no está enmascarada, el NDP causa una interrupción a la CPU, o bien directamente, o bien a través del Controlador Programable de

Interrupciones. La CPU para lo que está haciendo (y el NDP) y ejecuta la rutina apropiada de tratamiento del posible error. Esta rutina necesitará probablemente información del NDP; y a este efecto, hay ciertas instrucciones del NDP (como la FSTENV, FLDENV, FSAVE y FRSTOR) encargadas de salvaguardar y restaurar el estado actual del NDP incluyendo los contenidos de los ocho registros de datos. Son también muy útiles como ayuda en la determinación del tipo de error y el momento en que ocurrió. Detrás de casi cada paquete software del NDP debe haber un paquete de gestión de errores que asegure el buen funcionamiento. En el *Numeric Supplement to the 8086 Family User's Guide* de Intel hay ejemplos de estas rutinas.

PROGRAMAS-EJEMPLO

En esta sección veremos dos ejemplos, un programa de producto de matrices y un programa de cálculo de funciones trigonométricas. En este capítulo, no veremos los programas 8086/8088 correspondientes por ser bastante más largos. Por otro lado, el programa correspondiente usando el software del programa emulador del NDP 8087 tendría idéntico código fuente. Esto es, los mismos programas en lenguaje ensamblador llamarían a las rutinas software en vez de a las rutinas hardware, y ya hemos visto la diferencia de velocidad entre ambas.

El primer programa es una rutina de multiplicación de una matriz 4×4 por un vector de cuatro elementos. Esta rutina resulta clave para la potencia del NDP en la importante área del tratamiento de gráficos. Antes de ver la rutina, discutamos un poco cómo puede utilizarse. Supongamos que tenemos una escena tridimensional dibujada sólo con líneas rectas. Si se quiere pintar la escena en perspectiva, conviene representar cada punto por un vector de cuatro elementos; es decir, a cada punto se le asocian cuatro coordenadas. Esto se hace debido a que las *transformaciones proyectivas* que se necesitan para mostrar la escena desde varias perspectivas son fácilmente representables por el producto por una matriz 4×4. Más específicamente, para calcular la transformación proyectiva más general tomamos cualquier vector de tres elementos (x, y, z) y formamos un vector de cuatro añadiéndole 1 (x, y, z, 1). A continuación multiplicamos el vector por una matriz 4×4 apropiada, obteniendo el vector (x', y', z', w'). El vector de tres elementos correcto se obtiene dividiendo las cuatro coordenadas por w' y despreciando la última. La figura 4.14 muestra las fórmulas.

Matemáticamente hablando, lo más natural es representar los elementos de estas matrices y vectores por números en coma-flotante.

Además de la información sobre la posición, cada punto tiene una «quinta» coordenada para especificar cómo se conecta dicho punto con el previo. Un valor 0 significaría que no se dibuja línea desde el punto precedente; mientras que un valor 1 significaría una línea uniendo ambos puntos. Este es exactamente el código utilizado por la mayoría de los registradores digitales (plotters) para controlar si la pluma está levantada o no. Se pueden utilizar

MATRIZ: $\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$

VECTOR: $\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$

TRANSFORMACION:

$$x_1 \leftarrow \frac{a_{11}\, x_1 + a_{12}\, x_2 + a_{13}\, x_3 + a_{14}\, x_4}{a_{41}\, x_1 + a_{42}\, x_2 + a_{43}\, x_3 + a_{44}\, x_4}$$

$$x_2 \leftarrow \frac{a_{21}\, x_1 + a_{22}\, x_2 + a_{23}\, x_3 + a_{24}\, x_4}{a_{41}\, x_1 + a_{42}\, x_2 + a_{43}\, x_3 + a_{44}\, x_4}$$

$$\leftarrow \frac{a_{31}\, x_1 + a_{32}\, x_2 + a_{33}\, x_3 + a_{34}\, x_4}{a_{41}\, x_1 + a_{42}\, x_2 + a_{43}\, x_3 + a_{44}\, x_4}$$

Figura 4.14: Cálculo de la transformación proyectiva a través de un producto por una matriz.

códigos más elaborados, incluyendo la posibilidad de líneas de distintos colores. Los códigos de esas *conexiones* se representan normalmente por enteros. La figura 4.15 muestra cómo dibujar la letra A con esta notación:

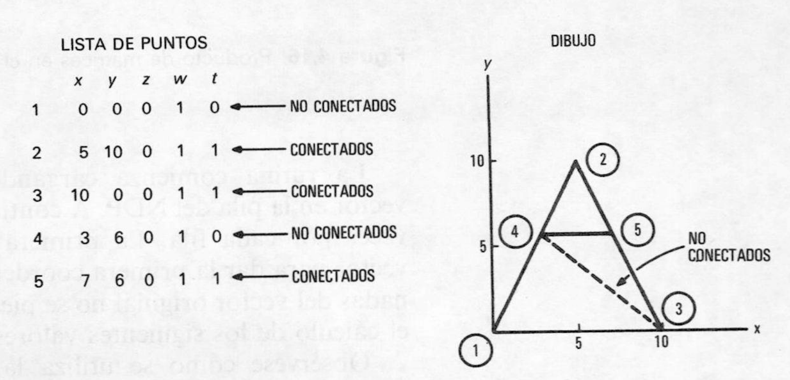


Figura 4.15: Dibujo de la letra A.

La figura 4.16 presenta una rutina del NDP 8087 en coma flotante que multiplica una matriz 4 × 4 por un vector de tamaño 4. Para cada elemento, el registro DI *apunta a* (contiene la dirección de) el vector de cuatro elementos original, y el BX *apunta a* la matriz 4 × 4. Los elementos de la matriz se almacenan fila a fila en posiciones consecutivas de memoria. El vector 4 resultante reemplaza al vector de cuatro elementos original.

```
                        ; PRODUCTO MATRICIAL 8087
                        ; ************************
                                NAME    MATPROD
                                PUBLIC  PROD
                        MAT     SEGMENT
                        ;
0004                    RSIZE   EQU     4
                                ASSUME  CS:MAT
                        ;
0000                    PROD    PROC    FAR
0000  1E                        PUSH    DS                              ; GUARDAR SEG. D
0001  55                        PUSH    BP                              ; GUARDAR PUNTERO BASE
0002  88  EC                    MOV     BP,SP                           ; APUNTA A LA PILA
0004  C5  7E  08                LDS     DI,DWORD PTR [BP+8]             ; OBTENER LA POSICION DEL VECTOR
0007  C5  5E  0C                LDS     BX,DWORD PTR [BP+12]            ; OBTENER LA POSICION DE LA MATRIZ
                        ;
000A  9B  DB  E3                FINIT                                   ; INICIALIZAR EL NDP
000D  9B  D9  45  0C            FLD     DWORD PTR [DI+3*RSIZE]          ; CARGAR X4
0011  9B  D9  45  08            FLD     DWORD PTR [DI+2*RSIZE]          CARGAR X3
0015  9B  D9  45  04            FLD     DWORD PTR [DI+1*RSIZE]          ; CARGAR X2
0019  9B  D9  05                FLD     DWORD PTR [DI]                  ; CARGAR X1
001C  B9  04  00                MOV     CX,4                            ; 4 FILAS
001F  9B  D9  07        ROW:    FLD     DWORD PTR [BX]                  ; OBTENER EL PRIMER ELEMENTO DE LA FILA
0022  9B  D8  C9                FMUL    ST,ST(1)                        ; MULTIPLICAR POR X1
0025  9B  D9  47  04            FLD     DWORD PTR [BX+1*RSIZE]          ; OBTENER EL SEGUNDO ELEMENTO DE LA FILA
0029  9B  D8  CB                FMUL    ST,ST(3)                        ; MULTIPLICAR POR X2
002C  9B  DE  C1                FADD                                    ; SUMARLO A LO ANTERIOR
002F  9B  D9  47  08            FLD     DWORD PTR [BX+2*RSIZE]          ; OBTENER EL TERCER ELEMENTO DE LA FILA
0033  9B  D8  CC                FMUL    ST,ST(4)                        ; MULTIPLICAR POR X3
0036  9B  DE  C1                FADD                                    ; SUMARLO A LO ANTERIOR
0039  9B  D9  47  0C            FLD     DWORD PTR [BX+3*RSIZE]          ; OBTENER EL CUARTO ELEMENTO DE LA FILA
003D  9B  D8  CD                FMUL    ST,ST(5)                        ; MULTIPLICAR POR X4
0040  9B  DE  C1                FADD                                    ; TOTAL
0043  9B  D9  1D                FSTP    DWORD PTR [D1]                  ; GUARDAR UNA ENTRADA DE Y
0046  83  C7  04                ADD     DI,1*RSIZE                      ; E INCREMENTAR EL PUNTERO
0049  83  C3  10                ADD     BX,4*RSIZE                      ; FILA SIGUIENTE
004C  E2  D1                    LOOP    ROW
004E  5D                        POP     BP                              ; RESTAURAR BP
004F  1F                        POP     DS                              ; RESTAURAR DS
0050  CA  08  00                RET     8                               ; VOLVER Y SALTAR 2 PUNTEROS
                        ;
                        PROD    ENDP                                    ; FIN DEL PROCEDIMIENTO
                        ;
                        MAT     ENDS                                    ; FIN DEL SEGMENTO
                                END                                     ; FIN DEL MODULO
```

Figura 4.16: Producto de matrices en el NDP 8087.

La rutina comienza cargando todos los componentes del vector en la pila del NDP. A continuación ejecuta un bucle cuatro veces por cada fila. La primera fila se «comprime» contra el vector para dar la primera coordenada del resultado. Las coordenadas del vector original no se pierden pues serán necesarias para el cálculo de los siguientes valores.

Obsérvese cómo se utiliza la pila para guardar resultados intermedios durante cada bucle. Si se examina el código máquina con detalle puede verse como cada operación en coma flotante está precedida por una instrucción de espera de CPU (incluso cuando no se codifica explícitamente en el código fuente). Es primordial para la correcta sincronización del NDP con la CPU.

El programa siguiente (véase la fig. 4.17), muestra el cálculo de las seis funciones trigonométricas. La instrucción clave es la FPTAN (tangente parcial). Es difícil encontrar esta instrucción en el programa porque se deben hacer muchas cosas con los datos y resultados antes y después de la operación mencionada.

Figura 4.17

```
                        ; FUNCIONES TRIGONOMETRICAS 8087
                        ; ******************************
                        NAME        TRIGMOD
                        PUBLIC      TRIG
                        TRIGSEG     SEGMENT
                                    ASSUME      CS:TRIGSEG,DS:TRIGDAT
                        ;
                        ;
0002                    C1          EQU         2
                        ;
0000                    TRIG        PROC        FAR
0000  1E                            PUSH        DS                  ; DS SE DEBE SALVAGUARDAR
0001  55                            PUSH        BP                  ; Y TAMBIEN BP
                        ;
0002  8B EC                         MOV         BP,SP               ; BP APUNTARA AL DATO
0004  83 C5 08                      ADD         BP,8                ; PERO PRIMERO SALTA DS, BP
                                                                    ; Y DEVUELVE LA DIRECCION CS, IP
                        ;
0007  B8 -- --                      MOV         AX,TRIGDAT
000A  8E D8                         MOV         DS,AX               ; TRIGDAT ES EL NUEVO DS

000C  9B D9 06 02 00    TRIGO;      FLD         NEGTWO              ; OBTENER-2
0011  9B D9 EB                      FLDPI                           ; OBTENER PI
0014  9B D9 FD                      FSCALE                          ; HACER PI/4
0017  9B D9 C2                      FLD         ST(2)               ; EL ANGULO ESTA EN EL ELEMENTO
                                                                    ; SUPERIOR DE LA PILA
                        ;
001A  9B D9 F8          TRIG1:      FPREM                           ; REDUCIR EL ANGULO
001D  9B DD 3E 00 00                FSTSW       STAT87              ; BITS DE ESTADO C3, C1, C0
0022  9B                            FWAIT                           ; DECIRLE QUE OCTANTE
0023  A1 00 00                      MOV         AX,STAT87
0026  9E                            SAHF                            ; ESTADO 87 EN INDICADORES 86
0027  7A F1                         JPE         TRIG1               ; HACER UN BUCLE HASTA QUE ESTE
                                                                    ; REDUCIDO TOTALMENTE
                        ;
0029  9B D9 F2                      FPTAN                           ; X=R*COS(ARG) Y Y=R*SEN(ARG)
                        ;
002C  73 0C                         JNC         TRIG2               ; PREGUNTAR POR C0
002E  9B D9 E0                      FCHS                            ; PARA LOS OCTANTES 4, 5, 6, 7
0031  9B D9 C9                      FXCH                            ; SE NECESITA CAMBIAR LOS DOS SIGNOS
0034  9B D9 E0                      FCHS
0037  9B D9 C9                      FXCH
                        ;
003A  75 06             TRIG2:      JNZ         TRIG3               ; PREGUNTAR POR C3
003C  9B D9 C9                      FXCH                            ; PARA LOS OCTANTES 2, 3, 6, 7
003F  9B D9 E0                      FCHS                            ; SE NECESITA CONMUTAR Y UN CAMBIO DE SIGNO
                        ;
0042  F6 C4 02          TRIG3:      TEST        AH,C1               ; C1 NO ESTA EN UN INDICADOR 86
0045  74 0F                         JZ          TRIG4               ; COMPROBARLO
0047  9B D9 C0                      FLD         ST                  ; PARA LOS OCTANTES 1, 3, 5, 7
004A  9B D9 C2                      FLD         ST(2)
004D  9B DE C1                      FADD                            ; SE NECESITA X-Y y X+Y
0050  9B D9 CA                      FXCH        ST(2)
0053  9B DE E9                      FSUB
;
                        TRIG4:                                      ; CALCULAR R
0056  9B D9 C0                      FLD         ST                  ; DUPLICAR X
0059  9B DC C8                      FMUL        ST,ST               ; ELEVARLO AL CUADRADO
005C  9B D9 C2                      FLD         ST(2)               ; OBTENER Y
005F  9B DC C8                      FMUL        ST,ST               ; ELEVARLO AL CUADRADO
0062  9B DE C1                      FADD                            ; AHORA TENEMOS X*X + Y*Y
0065  9B D9 FA                      FSQRT                           ; R = HIPOTENUSA
                        ;
0068  9B D9 C1                      FLD         ST(1)               ; OBTENER X
006B  9B D9 C1                      FLD         ST(1)               ; OBTENER R
006E  9B DE F9                      FDIV                            ; X/R
0071  C5 5E 14                      LDS         BX,DWORD PTR [BP+20]   ; DIRECCIONAR A
0074  9B D9 1F                      FSTP        DWORD PTR [BX]      ; GUARDAR EL COSENO
                        ;
0077  9B D9 C2                      FLD         ST(2)               ; OBTENER Y
007A  9B D9 C1                      FLD         ST(1)               ; OBTENER R
007D  9B DE F9                      FDIV                            ; Y/R
0080  C5 5E 10                      LDS         BX,DWORD PTR [BP+16]   ; DIRECCIONAR A
0083  9B D9 1F                      FSTP        DWORD PTR [BX]      ; GUARDAR EL SENO
                        ;
0086  9B D9 C2                      FLD         ST(2)               ; OBTENER Y
0089  9B D9 C2                      FLD         ST(2)               ; OBTENER X
008C  9B DE F9                      FDIV                            ; Y/X
008F  C5 5E 0C                      LDS         BX,DWORD PTR [BP+12]   ; DIRECCIONAR A
0092  9B D9 1F                      FSTP        DWORD PTR [BX]      ; GUARDAR LA TANGENTE
                        ;
```

```
0095 9B D9 C1              FLD     ST(1)                  ; OBTENER X
0098 9B D9 C1              FLD     ST(1)                  ; OBTENER R
009B 9B DE F1              FDIVR                          ; R/X
009E C5 5E 08              LDS     BX,DWORD PTR [BP+8]    ; DIRECCIONAR A
00A1 9B D9 1F              FSTP    DWORD PTR [BX]         ; GUARDAR LA SECANTE
                   ;
00A4 9B D9 C2              FLD     ST(2)                  ; OBTENER Y
00A7 9B D9 C1              FLD     ST(1)                  ; OBTENER R
00AA 9B DE F1              FDIVR                          ; R/Y
00AD C5 5E 04              LDS     BX,DWORD PTR [BP+4]    ; DIRECCIONAR A
00B0 9B D9 1F              FSTP    DWORD PTR (BX]         ; GUARDAR LA COSECANTE
                   ;
00B3 9B D9 C2              FLD     ST(2)                  ; OBTENER X
00B6 9B D9 C2              FLD     ST(2)                  ; OBTENER Y
00B9 9B DE F1              FDIVR                          ; X/Y
00BC C5 5E 00              LDS     BX,DWORD PTR [BP+0]    ; DIRECCIONAR A
00BF 9B D9 1F              FSTP    DWORD PTR [BX]         ; GUARDAR EL COTANGENTE
                   ;
00C2 9B DB E3              FINIT                          ; BORRAR LA FILA
                   ;
00C5 5D                    POP     BP                     ; RESTAURAR BP
00C6 1F                    POP     DS                     ; RESTAURAR DS
00C7 CA 18 00              RET     24                     ; SALTAR LOS DATOS
                   ;
                   TRIG    ENDP                           ; FIN DEL PROCEDIMIENTO
----               TRIGSEG ENDS                           ; FIN DEL SEGMENTO
                   ;
----               TRIGDAT SEGMENT                        ; SEGMENTO DE DATOS
0000 00 00         STAT87  DW      0
0002 00 00 00 C0   NEGTWO  DD      -2.
----               TRIGDAT ENDS                           ; FIN DEL SEGMENTO DE DATOS
                   ;
                           END                            ; FIN DEL MODULO
```

Figura 4.17: Programa NDP para la gestión de figuras trigonométricas.

Inicialmente, el ángulo está en el elemento superior de la pila del NDP. El primer paso consiste en reducir el ángulo al primer octante (entre 0 y pi/4) para poder aplicar la instrucción FPTAN. Esta reducción se hace a través de la instrucción FPREM (resto parcial), que, en este bucle, va restando pi/4 al ángulo hasta que entra en el primer octante. El proceso para varias veces, sin haberse completado, para que pueda procesarse cualquier interrupción de la CPU. El fin de la instrucción puede detectarse mirando un bit particular de la palabra de control del NDP. Este bit se examina cargando la mitad superior de la palabra de control en el octeto superior del registro de indicadores de la CPU. El bit *proceso-acabado* se hace uno a la vez que el bit de paridad, de manera que el programa hace un bucle hasta que el indicador de paridad le dice que puede continuar con el resto del programa.

La instrucción FPTAN devuelve dos valores X e Y que coloca en los dos elementos superiores de la pila del NDP. El paso siguiente es el de ajustar estos dos valores para los distintos *octantes*. Es decir, pi/4 divide al círculo entero en ocho octantes, de forma que las funciones trigonométricas son combinaciones algebraicas de X e Y distintas para cada octante. En nuestro programa, incluso se ajusta el ángulo como parte del proceso. El octante correcto se determina mediante ciertos bits de la palabra de control del NDP que se carga en los indicadores de la CPU.

Una vez ajustados X e Y, se calcula R=SQRT(X*X+Y*Y). R es la hipotenusa de un triángulo rectángulo de catetos igual a X e Y.

La última parte del programa calcula diversas razones de X, Y y R que nos dan las seis funciones trigonométricas. Estos valores se almacenan en seis posiciones distintas y se introducen en la pila del NDP para devolver el resultado.

CONCLUSIONES

Como acabamos de ver, el NDP 8087 amplía en gran medida las posibilidades del 8086/8088, elevando su rendimiento al nivel de las minicomputadoras de la generación anterior (todavía en activo en muchas instalaciones), y proporcionando una precisión incluso mayor que las acostumbradas en grandes sistemas.

El acoplar procesadores en paralelo al sistema es una solución al problema de la expansión de las posibilidades de la CPU, siempre que se conserve una compatibilidad software (y hasta cierto punto hardware). Siguiendo esta idea, sería posible acoplar otros chips además del NDP 8087 (incluso chips de otros fabricantes) que ampliaran el conjunto de instrucciones del 8086/8088 en otras áreas, aumentando su eficiencia en ellas.

Capítulo 5

El procesador de entrada/salida 8089

La figura 5.1 muestra la estructura interna y los terminales del Procesador de Entrada/Salida 8089. Se trata éste de un microprocesador encargado de gestionar la entrada/salida para la CPU 8086/8088. El IOP 8089 ejecuta su propio flujo de instrucciones, comunicándose con la CPU 8086/8088 a través de unas pocas líneas de señal y de grandes bloques de información guardados en la memoria del sistema y/o en el espacio de E/S del sistema. En el capítulo 2 ya vimos cómo conectar el IOP 8089 en un sistema de computadora completo. Vimos, de hecho, un sistema basado en el 8086 de 16 bits y otro basado en el 8088 de 8 bits, y ¡ambos utilizaban la única versión existente del IOP 8089! Antes de estudiar *cómo* trabaja el IOP 8089, vamos a discutir *por qué* es necesario.

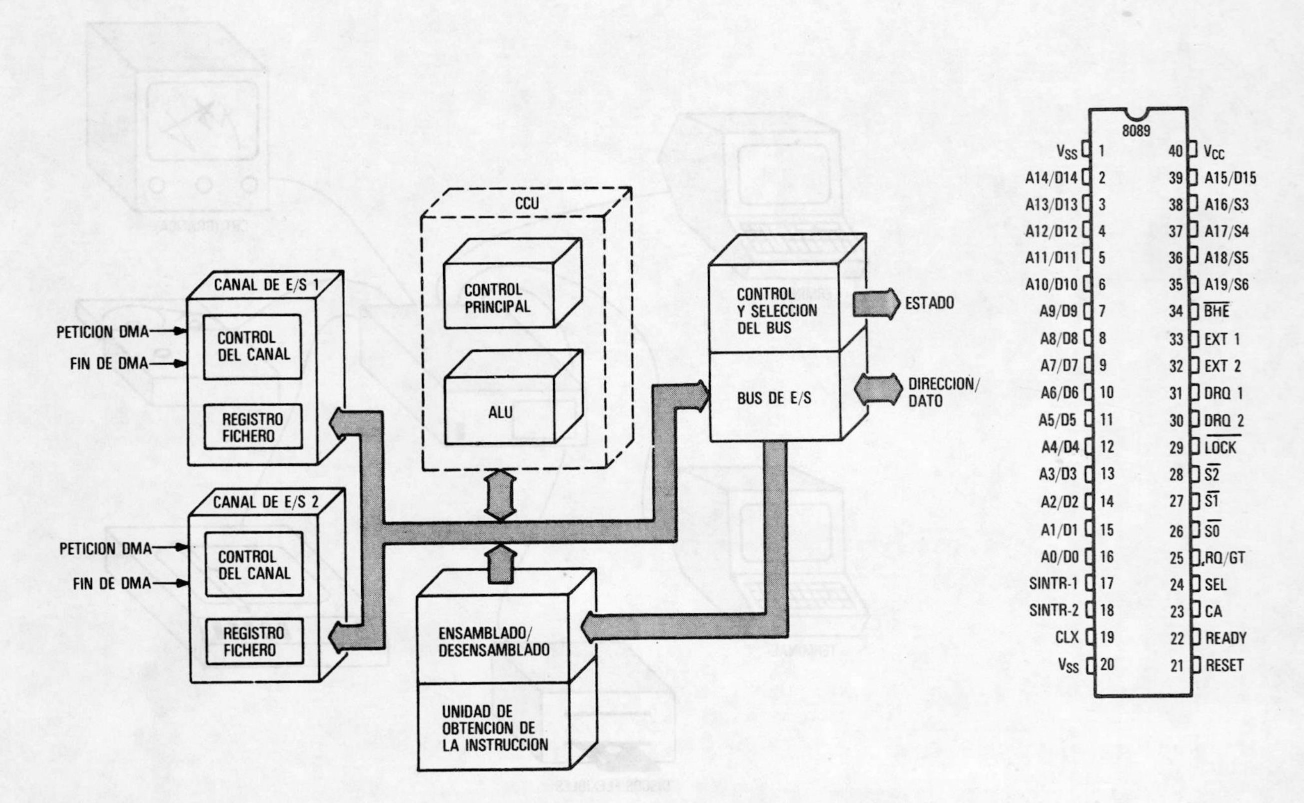


Figura 5.1: Estructura interna y terminales del IOP 8089.

ENTRADA/SALIDA

Así como las aplicaciones de los microprocesadores han evolucionado desde su uso en calculadoras, a terminales, y finalmente a sistemas computadoras completos, también la gestión de la E/S de Intel se ha hecho cada vez más sofisticada. Para una calculadora, los programas son cortos y consisten normalmente en evaluaciones de expresiones introducidas vía el teclado de la máquina, visualizadas sobre una pequeña pantalla de pocos caracteres. Por el contrario, un sistema de microcomputadora moderna ejecuta complejos y largos programas que pueden necesitar fácilmente de media docena de distintos dispositivos de E/S. Estos dispositivos pueden incluir teclados, discos flexibles, unidades de cinta, visualizadores de textos, visualizadores de gráficos, modems, impresoras, mesas digitalizadoras y registradores gráficos digitales (plotters). En la figura 5.2 puede verse la evolución habida, desde las calculadoras a las microcomputadoras actuales.

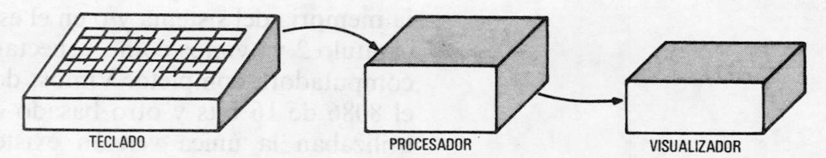


Figura 5.2a: E/S para una calculadora.

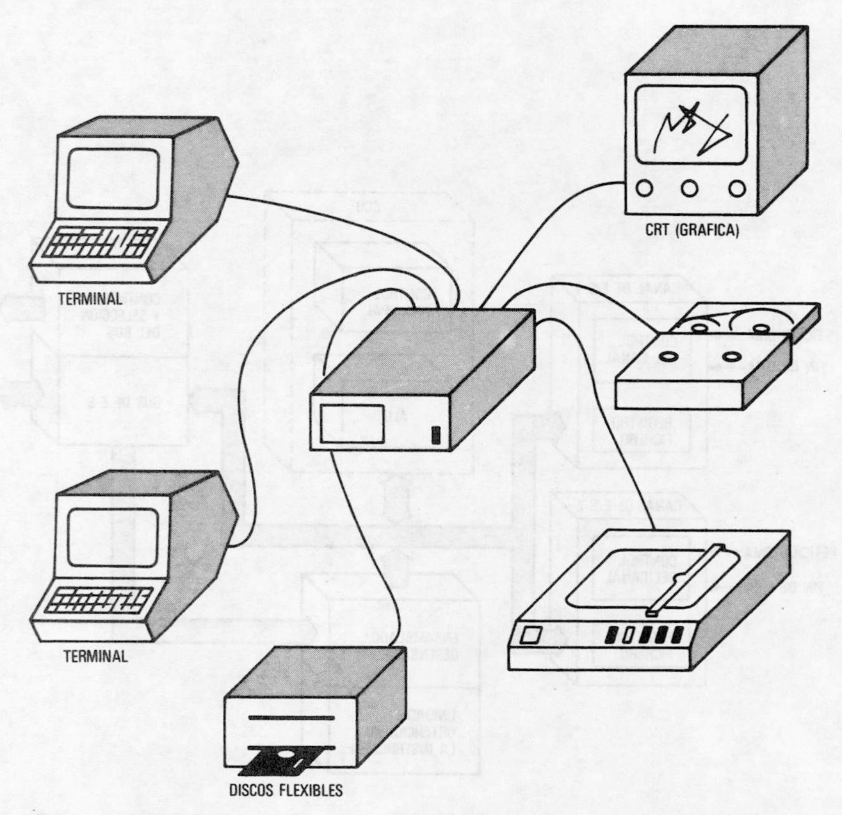


Figura 5.2b: E/S para una computadora.

Cinco puntos importantes Cuando se habla de la E/S, hay varias consideraciones a tener en cuenta: la velocidad de transmisión, la forma en la que se agrupa la información, el grado de control necesario, la cantidad de traducción o conversión de los datos necesaria y el sistema de detección y corrección de errores utilizado. Cada dispositivo de E/S tiene requerimientos distintos con respecto a cada uno de estos puntos.

1. La velocidad de transmisión de la información de un cierto dispositivo de E/S puede variar desde ser varios órdenes de magnitud más lenta que la CPU, a ser incluso más rápido que éste. Por ejemplo, si una CPU está controlando directamente un teclado, ¡tendrá que esperar millones de ciclos de reloj para que se transmita un solo carácter! La CPU perdería más del 99,99 por 100 de su tiempo esperando a que el operador entrase la letra siguiente. Entonces, una vez la orden se hubiese completado, el operador debería esperar a que la CPU hiciese su trabajo. Como se ve, el sistema sería una pérdida de tiempo tanto para la CPU como para el operador. Cuando uno estuviera trabajando, el otro necesariamente perdería el tiempo. En el extremo opuesto, si la CPU controlase directamente la visualización de la información sobre una pantalla, el texto sobre la pantalla aparecería «perezoso», al no poder la CPU obtener y enviar la información con la suficiente rapidez.

2. Los datos se suelen agrupar en paquetes de varios tamaños, a menudo en paquetes formados a su vez por paquetes menores. La unidad básica de información es el bit (dígito binario). Cierto número de bits (de 4 a 64) se agrupan para formar una *palabra,* con el caso especial de *octeto,* que son 8 bits. Finalmente un cierto número de palabras (de 0 a varios miles) se agrupan y forman un *bloque*. La figura 5.3 muestra estos agrupamientos. Por ejemplo, un bit puede utilizarse para controlar la abertura o no de una cierta válvula en un coche controlado por microprocesador, un

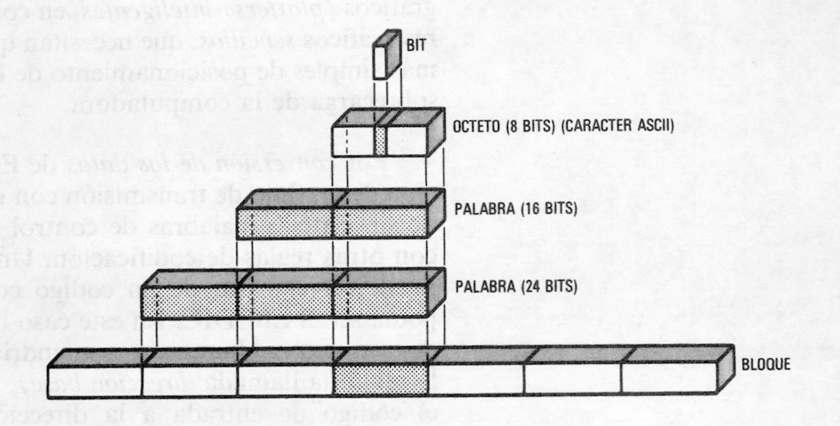


Figura 5.3: Agrupaciones de datos.

octeto puede enviarse a un elemento visualizador de una calculadora para formar un dígito; una pantalla completa (24 líneas × 80 columnas) de caracteres (octetos) puede enviarse al terminal de un procesador de textos; o, se puede escribir un gran fichero en disco que incluya miles de bloques, cada bloque de 128 octetos. Esta amplia variación es cierta tanto para entrada como para salida.

3. Por *control* de la E/S entendemos la gestión programada de los dispositivos de entrada/salida que posibilita la conversión de las señales eléctricas provenientes de la computadora en señales entendibles por el operador humano y viceversa. Se puede imaginar un flujo de transmisión como un programa que, cuando se ejecuta, causa un efecto concreto deseado, como puede ser la impresión de una página de un texto; la visualización de una cierta información sobre una pantalla, o el dibujo de una gráfica en un plotter. Normalmente, un flujo de transmisión consta de datos y señales de control intermezcladas. Las palabras de control actúan a modo de «instrucciones» mezclándose con los «datos» que es la información que se está transmitiendo. Por ejemplo, en la transmisión de un texto, la información sobre el control de carro, salto de línea, o el fin de un mensaje se envía como *octetos de control,* intercalados con los octetos de los caracteres del texto. Dependiendo del dispositivo de salida particular los octetos de control harán que se realicen distintas acciones físicas. Así, una vuelta de carro se ejecutará de distinta manera en una impresora que en una pantalla alfanumérica. La cantidad de trabajo necesaria para que la computadora «ejecute» estos «programas» también varía de un dispositivo a otro. No puede costar lo mismo enviar caracteres ASCII uno a uno a través de un port de salida, que ejecutar rutinas de dibujo de complicadas líneas para un plotter digital. Gran parte de los dispositivos de E/S actuales tienen sus propios microprocesadores que realizan todas estas tareas. Por ejemplo, algunos plotters digitales ejecutan sus propias rutinas de dibujo como respuesta a órdenes sencillas provenientes del procesador central. Son los llamados registradores gráficos *(plotters) inteligentes,* en contraposición a los registradores gráficos *sencillos,* que necesitan que la CPU controle sus pasos más simples de posicionamiento de la pluma, con la consiguiente sobrecarga de la computadora.

4. Por *conversión de los datos* de E/S se entiende la transformación de un flujo de transmisión con un conjunto de codificaciones de sus datos y palabras de control en otro flujo de transmisión con otras reglas de codificación. Una conversión de datos típica puede ser el pasar de un código como el ASCII a otro, como podía ser el EBCDIC. En este caso bastaría con una simple tabla de conversión. Un registro contendría la dirección de comienzo de la tabla (la llamada *dirección base),* de modo que bastaría sumar el código de entrada a la dirección base para determinar la posición del código traducido dentro de la tabla.

5. Por esquema de *detección y corrección de errores* significamos aquellos métodos de envío de la información digital por líneas de transmisión con posibles ruidos, que aseguren que cualquier error introducido durante la transmisión puede ser eliminado, o al menos detectado a su llegada al punto de destino. Estos esquemas pueden ser desde un «cruzar los dedos» y esperar que no suceda nada, hasta elaborados métodos de control de paridad.

Típicamente, se añade una información extra al mensaje antes de ser enviado y esta información se utiliza en el punto de destino para detectar y/o corregir los posibles errores de transmisión.

Por ejemplo, en los esquemas de control de paridad hay unos bits especiales llamados *bits de paridad* que se añaden a cada palabra justo antes de transmitirla. La palabra resultante recibe el nombre de *palabra código*. Estas palabras código son las que se envían por la línea. La detección de errores se basa en el hecho de que no todas las palabras son en realidad palabras código. Por ejemplo, si añadimos un bit extra de paridad a un código ASCII de 7 bits (siguiendo la regla tan sencilla de que la suma de unos de la palabra código resultante sea siempre par, o siempre impar), entonces las palabras código resultantes tienen 8 bits, pero no todas las palabras de 8 bits son palabras código. Obsérvese que hay 128 palabras de código ASCII de 7 bits, y 256 de 8 bits. Sólo la mitad de las palabras de 8 bits pueden ser palabras código como las definidas anteriormente. Cuando en el punto de destino se recibe una palabra se comprueba si es una palabra código (en nuestro caso, si el número de dígitos es par, o impar). Si no lo es, es que debe haberse introducido un error durante la transmisión y ésta debe repetirse.

Si se usan más bits que un único bit de paridad, se pueden enviar más datos sobre la palabra original (carácter), de modo que pueda corregirse el error. Tales esquemas de paridad pueden detectar sólo un número pequeño de bits erróneos en la palabra. Por ejemplo, con un solo bit de paridad se pueden detectar únicamente errores de un solo bit mal transmitido, y *no* se puede corregir ningún error. Si durante la transmisión se introduce errores en dos bits, en el punto de destino se recibirá también una palabra código, pero no la correcta, y no habrá manera de detectarlo.

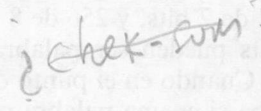
Un esquema de detección y corrección de errores muy popular y que utiliza varios bits de paridad es el llamado código de Hamming, debido a Richard Hamming, actualmente en la Naval Postgraduate School de Monterrey, California (ver Hamming, R. W. *Coding and information theory,* Englewood Cliffs, N. J.: Prentice Hall, 1980). Con el código de Hamming, las palabras del mensaje original (el texto) son de 4 bits y se les añaden 3 bits de paridad, dando palabras código de 7 bits. Con este esquema se pueden detectar errores *dobles* (2 bits erróneos) y corregir errores *simples* (un bit erróneo).

La corrección de errores se realiza seleccionando la palabra código *más cercana* a la palabra recibida. La «más cercana» se

mide en términos de la que está a menor distancia de, definiendo la distancia entre dos palabras *(distancia de Hamming)* como el número de dígitos que, en la misma posición relativa, tienen valores distintos en las dos palabras. Por ejemplo, la distancia entre la palabra 1010111 y la 1110110 es dos puestos que difieren en exactamente dos posiciones. En el código de Hamming de 7 bits, cualquier palabra de 7 bits, o bien es una palabra código, o bien está a distancia *uno* de una única palabra código. Dicho de otra manera, para *cualquier* palabra de 7 bits *recibida,* hay una y *sólo una* palabra código más cercana (que se diferencia de la recibida en un solo bit). Esta palabra código es la que se escoge durante el proceso de corrección.

Como puede verse, los esquemas de control de paridad son buenos, pero distan de ser perfectos. Si consideramos que es mucho más probable que se introduzcan un número pequeño de errores por palabra que un número grande, estos esquemas parecen suficientemente fiables.

Otro método de detección de errores muy utilizado en cintas y discos flexibles es el de *verificación de redundancia cíclica.* En este método se utiliza un octeto *verificador* especial al final de cada sector de datos. Si esto no se verifica, el sector es erróneo. Puede haber una secuencia completa de errores que se anulen entre sí. El octeto verificador se calcula sumando todos los octetos del mensaje y guardando únicamente el octeto menos significativo de la suma.



Arquitectura orientada a bus

El primer paso para gestionar los diversos dispositivos de E/S y sus necesidades particulares es el tener un bus control al cual está conectado cada dispositivo a través de su controlador propio. La figura 5.4 muestra un esquema de esta estructura. Esta es la

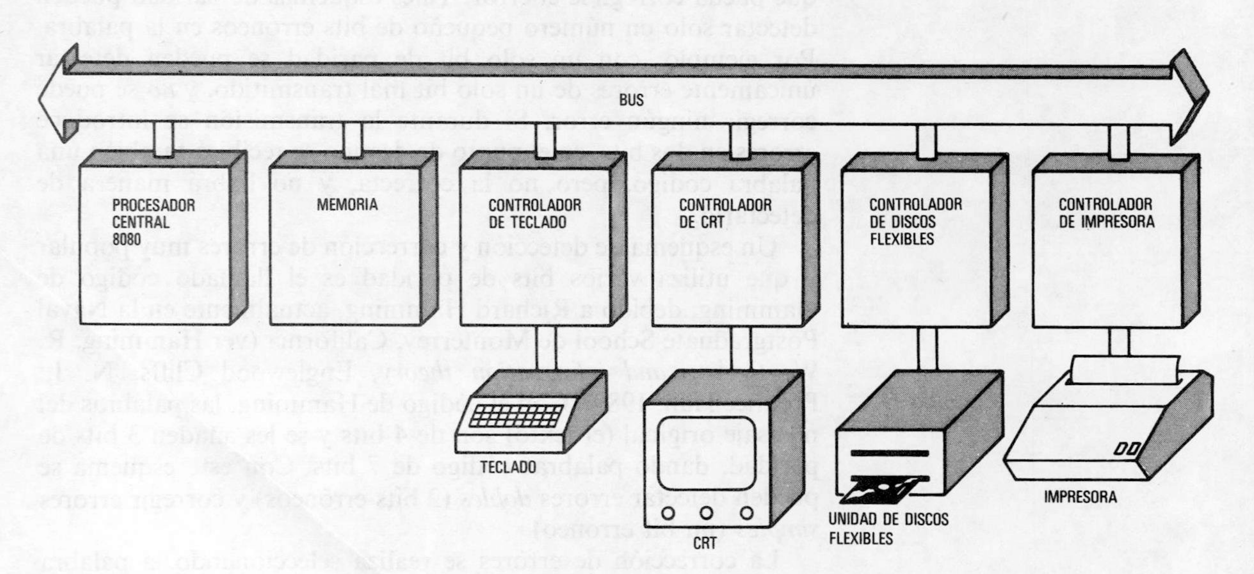


Figura 5.4: Bus de E/S.

filosofía de la arquitectura basada en buses de Intel y de sus chips controladores de dispositivos como la Interfaz Programable para Teclado 8278, y el Controlador Programable de CRT 8275.

E/S por sondeo periódico y por interrupción

En un sistema de bus como el descrito en el apartado anterior, existen normalmente dos formas de gestionar la E/S: por sondeo periódico y por interrupción. El problema fundamental con cualquier operación de E/S es la tremenda diferencia de velocidad entre el procesador y los dispositivos de entrada-salida. Por ejemplo, el 8086 puede trabajar a 5 MHz, o lo que es lo mismo, procesa unas 100.000 instrucciones por segundo. Entrando los datos por teclado, en condiciones normales se suelen teclear unos pocos caracteres por segundo. Una línea telefónica puede transferir como mucho 120 caracteres por segundo (típicamente la velocidad varía de 960 caracteres por segundo si se utilizan líneas especiales, a 11 ó 30 caracteres por segundo con un modem barato).

Con el sistema de E/S por sondeo periódico, el procesador debe esperar la transmisión de cada carácter antes de que pueda ocuparse de otras tareas. Mientras espera, el procesador ejecuta un bucle en el que pregunta continuamente por un octeto de E/S especial llamado registro de estado. Este registro de estado se halla conectado a la circuitería del controlador del dispositivo en la computadora. Cuando dicha circuitería está preparada para la transferencia (sea porque el dispositivo externo está preparado, sea debido a la sincronización en la circuitería del computador), envía una señal de «preparado» al port de estado donde la puede ver la CPU. La CPU entonces hace la transferencia, y sólo entonces abandona el bucle y comienza otras tareas.

Por el contrario, con el sistema más efectivo de interrupciones, el procesador continúa trabajando, *sin* preguntar aparentemente por el dispositivo de E/S, al menos por software. *Incorporado en el hardware hay un verificador de señales de los controladores de dispositivos de E/S.* Cuando el controlador de un dispositivo de E/S indica que éste está preparado para la transferencia, envía una señal de interrupción o bien directamente a la CPU, al Controlador Programable de Interrupciones (PIC 8259) que se lo indica a la CPU. El procesador acaba la instrucción en curso, salvaguarda la suficiente información para poder luego continuar por el mismo punto, y bifurca a una rutina especial de *tratamiento de interrupciones* encargada de realizar la transferencia. La transferencia se suele realizar entre el dispositivo y un área especial que recibe el nombre de almacenamiento temporal (buffer). Acabada la transferencia, el procesador vuelve (gracias a la información salvaguardada) al punto del proceso que estaba ejecutando cuando llegó la interrupción. El procesador puede acceder a los datos del almacenamiento temporal cuando los necesite. De esta manera, la transferencia de información carácter a carácter se realiza concurrentemente con otras tareas. Desde luego, el procesador acabará eventualmente todos los otros

trabajos y deberá esperar nueva información de los dispositivos externos antes de continuar. El método de las interrupciones es particularmente útil y apropiado en casos en los que el procesador gestiona varias tareas a la vez. Por ejemplo, en un sistema de tiempo compartido, mientras el programa de un usuario está esperando la siguiente línea de órdenes, o imprimiendo resultados, los de otros usuarios pueden estar simultáneamente calculando valores. Nosotros utilizaremos este método de las interrupciones de E/S en nuestro modelo de computador. La figura 5.5 muestra un esquema de ambos métodos.

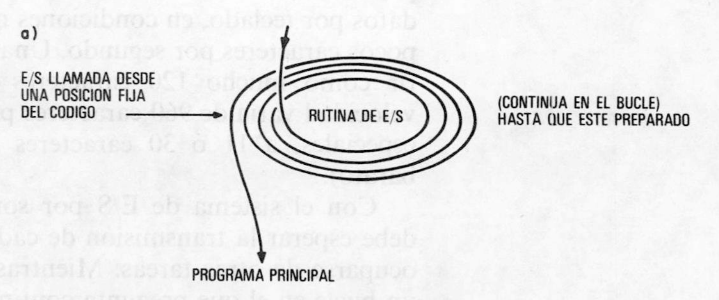


Figura 5.5a: E/S por sondeo periódico.

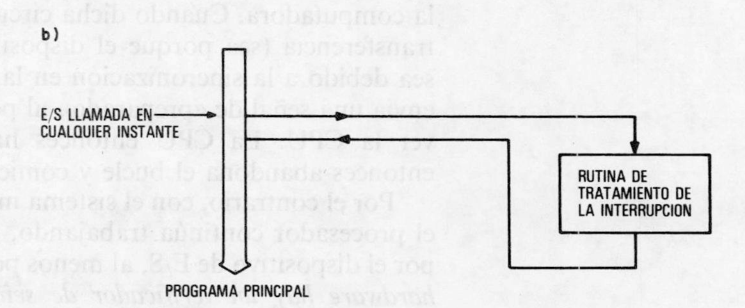


Figura 5.5b: E/S por interrupciones.

Algunas interrupciones son más importantes que otras. Por ejemplo, el teclear un carácter debe procesarse tan pronto como sea posible, mientras que la transferencia de un carácter a una impresora normalmente puede esperar una fracción de segundo sin demasiadas dificultades. Por lo tanto, a veces resulta interesante asignar prioridades a las interrupciones provenientes de dispositivos distintos, de modo que las más importantes se atiendan primero.

El uso de interrupciones de E/S tiene algunas desventajas. La CPU todavía necesita encargarse de la transmisión de información octeto a octeto. Cada vez que se transmite un nuevo octeto, el procesador debe parar lo que está haciendo, salvaguardar sus registros de trabajo, hacer la transferencia, restaurar sus registros

de nuevo y retomar la tarea que había abandonado. Esto puede suceder en cualquier punto del programa (excepto en las secciones en las que se hayan enmascarado las interrupciones), lo que puede sobrecargar considerablemente al procesador, y por tanto ocasionar una pérdida notable de velocidad. Hay ciertas operaciones críticas, como la E/S a disco, que requieren un servicio interrumpido de la CPU. Para poder atender estos casos se desactivan a propósito las interrupciones en las secciones del código que gestionan dichas operaciones. Mientras se están ejecutando esas secciones, las entradas de cualquier otro dispositivo se ignoran. Este punto resulta crucial para el buen funcionamiento de la computadora. Otro problema que puede surgir es que las interrupciones sucedan tan a menudo (por ejemplo, cada barrido de línea —63 microsegundos—) que la CPU no tenga tiempo de procesarlas.

Para solucionar algunos de estos problemas, se puede añadir un procesador que *se encargue* exclusivamente de la E/S, bajo la *dirección* de la CPU. De hecho sería incluso más práctico tener un procesador encargado de la E/S de cada dispositivo periférico, o al menos de cada tipo de dispositivo.

En esta línea, Intel ha desarrollado el procesador de E/S 8089; un procesador de propósito especial con un juego de instrucciones especialmente pensado para una eficiente gestión de las operaciones de E/S. Tiene dos canales distintos de E/S que pueden ofrecer servicio a dos periféricos de E/S simultáneamente e incluso posibilita la existencia de otros I/O acoplados a otros dispositivos de E/S.

PROCESADOR PERIFERICO Y MULTIPROCESO

El IOP 8089 es un procesador por sí mismo, pero con un juego de instrucciones de propósito especial y un procedimiento de inicialización muy particular. Pertenece a la nueva generación de microprocesadores para procesos paralelos y multiprocesos, actuando a modo de *procesador periférico*. El IOP 8089 está diseñado para ser utilizado como *intermediario* para el 8086 o el 8088. Es decir, todos los dispositivos de E/S se comunican con el 8089 que a su vez se comunica con el computador central. Esta actuación en equipo aumenta sustancialmente el rendimiento de todo el sistema. El IOP 8089 será el que tenga que esperar a los dispositivos de E/S (como teclados, etc.), quedando liberada de esta tarea la CPU, que se podrá encargar de realizar los cálculos generales y de gestión a todo el sistema. El IOP 8089 es capaz de gestionar la E/S mejor que la CPU al poder transferir datos entre la memoria y los dispositivos de E/S con varios tipos de conversión, y controlar la E/S en menos ciclos de los que necesitaría el 8086/8088. Este uso de un dispositivo como «intermediario» es bastante común en los grandes computadores, e incluso a veces utilizan minicomputadores como el PDP-11 a tal efecto, como es el caso del Cray.

En la figura 5.6, dos IOP 8089 hacen de intermediarios para un 8086. Cada IOP tiene dos canales, uno por dispositivo de E/S. Cada canal se comunica directamente con el controlador del

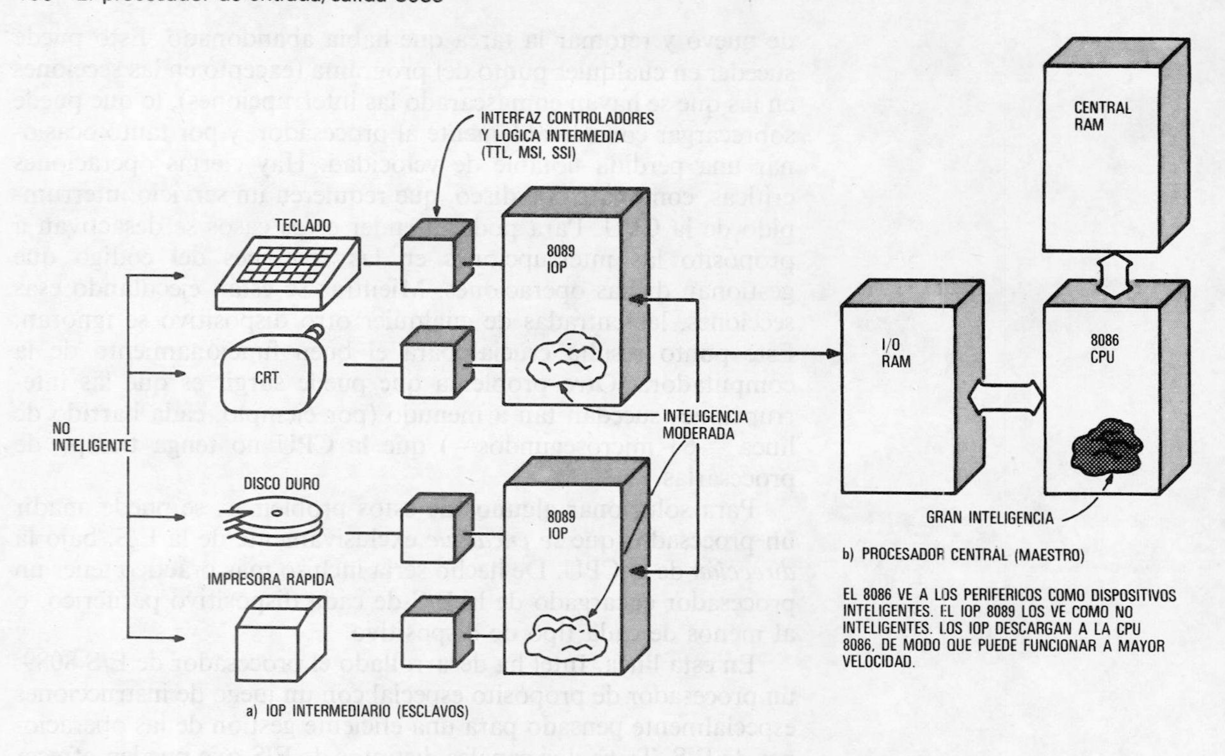


Figura 5.6: Control de la E/S con el IOP 8089.

dispositivo correspondiente. El IOP de arriba se encarga de un teclado por el canal 1 y de una CRT por el canal 2, mientras que el IOP inferior gestiona una unidad de disco por el canal 1 y una impresora rápida por el canal 2. Ambos IOP usan el espacio de E/S del 8086 para almacenar sus datos y programas. El 8086 dirige las actividades de los dos IOP leyendo y escribiendo la información de/en los bloques del espacio de E/S. La RAM central se reserva para la CPU 8086. No se muestra en el dibujo la estructura de buses del sistema. En el capítulo 2 se mostraba tal estructura.

En los sistemas que incluyen un IOP, la CPU (8086 u 8088) supervisa todo el sistema. Envía órdenes al IOP 8089 de dos maneras: 1) a través de las líneas de señal que unen ambos procesadores, y 2) mediante mensajes en zonas de memoria compartida. Hay varias líneas de señal entre la CPU y el IOP 8089. Algunas de estas líneas de señal (las de petición/concesión, RQ/GT), posibilitan el que la CPU y el IOP compartan eficientemente el mismo bus. Estas señales funcionan de la siguiente manera: Hay una línea de señal RQ/GT que une el IOP 8089 con la CPU. Cuando el IOP necesita utilizar el bus envía una señal de petición por dicha línea. Tan pronto como la CPU abandona el bus, desconecta eléctricamente (estado de alta impedancia) sus terminales, y envía un pulso de vuelta a la línea RQ/GT. El IOP

utiliza el bus, conectando eléctricamente sus líneas, y realiza las transferencias. Cuando acaba, desconecta sus líneas de bus y devuelve un pulso a la CPU (por la misma línea RQ/GT). Una vez recibido el pulso de «abandono» del bus por el IOP, la CPU reinicia su actividad normal sobre el bus.

Hay una línea (la línea de atención del canal) que permite a la CPU dar un rápido «puntapié» al IOP 8089 informándole de que no hay trabajo a la vista. Los detalles exactos de qué trabajo hay que hacer y cómo los carga previamente la CPU en ciertos bloques especiales de memoria.

Como respuesta a estas órdenes, el IOP 8089 realiza las tareas de E/S correspondientes por medio de su limitado juego de instrucciones. Dichas tareas incluyen la inicialización y supervisión de los controladores de dispositivo, proporcionándoles las instrucciones necesarias para que éste pueda transferir una unidad de información completa entre los dispositivos de E/S y la memoria del ordenador. La operación clave es la instrucción XFER que realiza transferencias inteligentes, similares a la DMA (veremos esto más tarde) entre los dispositivos de E/S y la memoria. Estas transferencias, aunque se realizan más rápidamente de lo que puede hacerlo la CPU, permiten el mismo tipo de control y conversión de datos que el que realizaría la CPU. De todas las instrucciones de la CPU, la operación de transferencia al 8089 es la más similar a una operación general de tratamiento de cadenas. De hecho, una de las utilidades del IOP puede ser el de realizar movimientos de bloques y operaciones sobre cadenas para la CPU, como hacíamos en nuestro modelo de computadora. Una vez el IOP 8089 ha hecho su trabajo, todos los dispositivos de E/S aparecen iguales desde el punto de vista de la CPU; más concretamente, todos ellos aparecen como un bloque de memoria. De hecho, podemos imaginarnos cada bloque como un fichero que, igual que con un fichero en disco, puede abrirse, leerse, escribirse y cerrarse.

MODO LOCAL Y MODO REMOTO

El 8089 puede conectarse a un sistema computador básicamente de dos maneras: en modo *local* y en modo *remoto*. En modo *local* la CPU, IOP, los controladores de dispositivos y la memoria central se hallan conectados al bus principal del sistema. Los modelos que se ven en este libro están en modo local. La CPU 8086/8088 actúa a modo de maestro y el IOP 8089 como esclavo. Las líneas de señal conectan una terminal de la línea de petición/concesión (RQ/GT) de la CPU con uno de los dos terminales correspondientes del IOP. Como ya hemos visto, la línea de petición/concesión es una línea bidireccional gracias a la cual se determina cuál de los dos procesadores tiene el control del bus. La ventaja del modo local reside en su bajo costo, al requerir un número reducido de chips; pero tiene como desventaja la carga que añade al bus. Es decir, cualquier bus (igual que una autopista) puede absorver una cierta cantidad de tráfico antes de colapsarse (en una hora punta). La figura 5.7 muestra la configuración local.

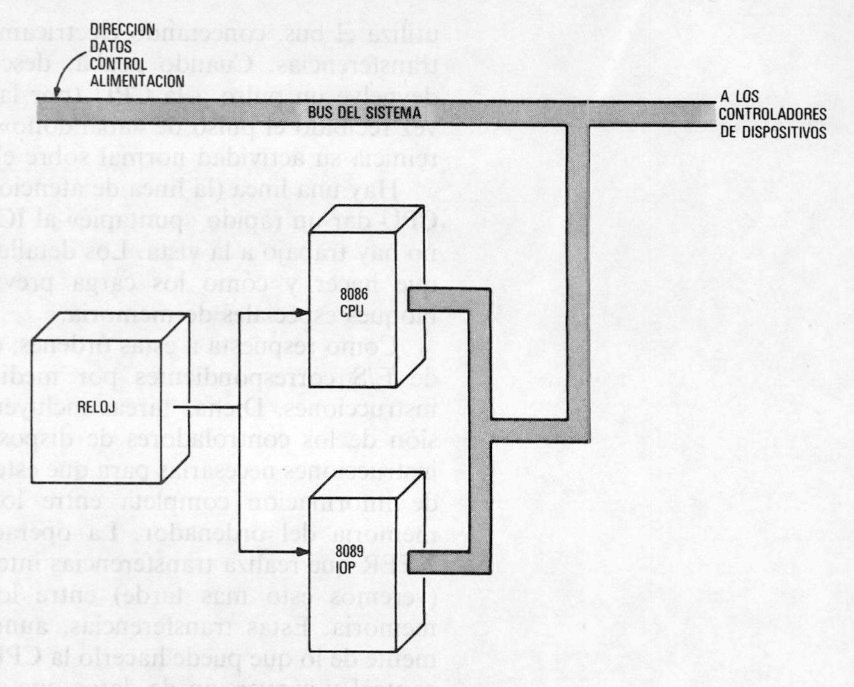


Figura 5.7: Modo local para el IOP 8089.

En el modo remoto, uno de los dos IOP forma un pequeño grupo junto con su propio bus local. Los controladores de E/S se conectan a dicho bus. Hay además un bus del sistema al cual se conecta este grupo. En un sistema grande, podría haber varios grupos de manera que la actividad de E/S se distribuiría entre varios procesadores y varios buses distintos. Por ejemplo, cada usuario podría tener un grupo diferente, y la impresora y las unidades de disco del sistema podrían tener acceso a otro.

El IOP 8089 tiene las mismas necesidades de sincronización y alimentación que el 8086/8088. Puede programarse (mediante un octeto especial de memoria que se lee durante la inicialización)

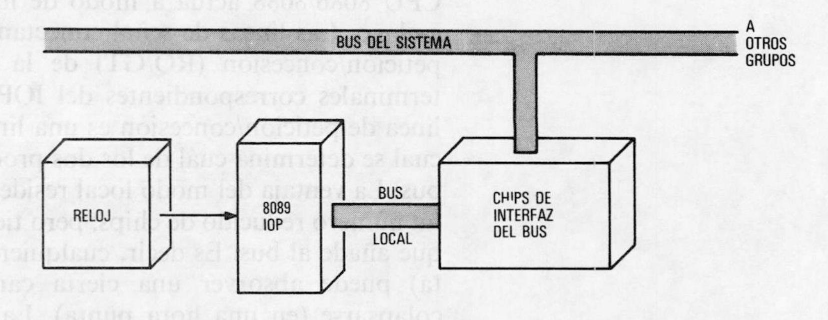


Figura 5.8a: Modo remoto para el IOP 8089 (un IOP en un grupo).

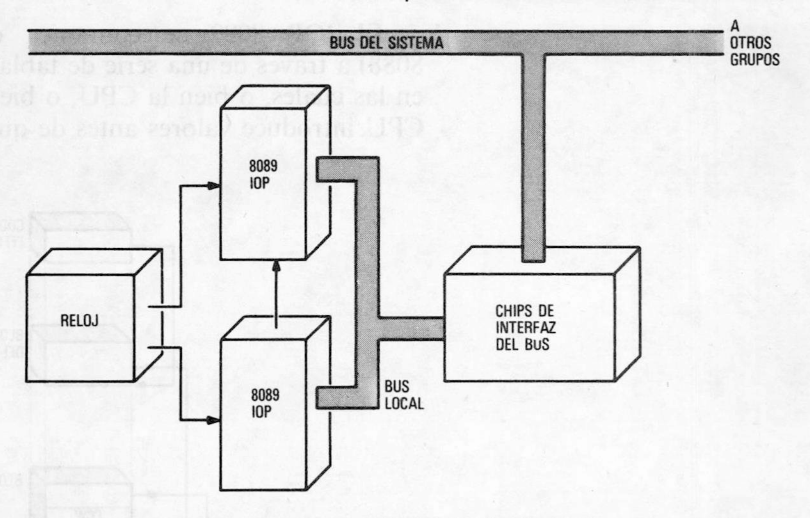


Figura 5.8b: Modo remoto para el IOP (8089) (dos IOP en un grupo).

para que presente un bus de datos de 8 bits como el 8088, o de 16 bits como el 8086.

COMO TRABAJA EL IOP 8089

El IOP 8089 es capaz de procesar dos trabajos (llamados *tareas)* al mismo tiempo. Cada trabajo podría ser la interfaz de un dispositivo de E/S distinto. Varios trabajos que se realizan a la vez reciben el nombre de procesos concurrentes. Para poder ejecutar dos trabajos concurrentemente, el IOP 8089 está dividido en dos secciones llamadas canales, uno para cada tarea. Cada canal tiene su propio juego de registros y su propio programa, al que se da el nombre de *bloque de tarea* y que reside o bien en la memoria del sistema, o en el espacio de E/S. Sin embargo, en un momento dado, realmente sólo un canal está realizando tareas. La Unidad Común de Control (CCU) se encarga de coordinar las tareas de los dos canales, dando control al canal de prioridad más alta o, si ambos tienen la misma prioridad, solapando ambas tareas de manera que se realicen alternativamente instrucciones de ambos canales (véase la figura 5.9).

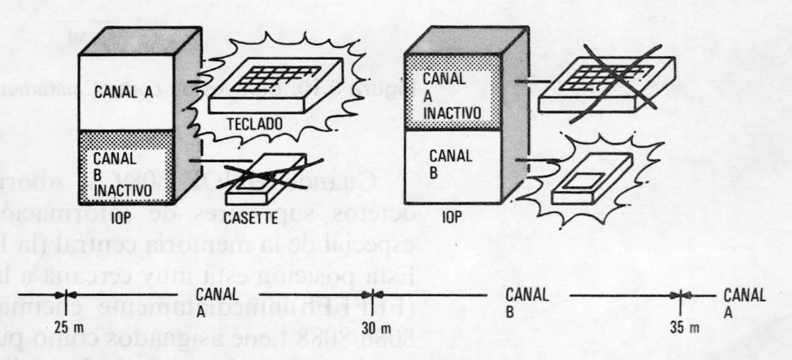


Figura 5.9: Solapamiento de los dos canales del IOP 8089.

El IOP 8089 se comunica con el procesador (8086 u 8088) a través de una serie de tablas llamadas bloques (fig. 5.10), en las cuales, o bien la CPU, o bien otro IOP bajo control de la CPU introduce valores antes de que el IOP comience a trabajar.

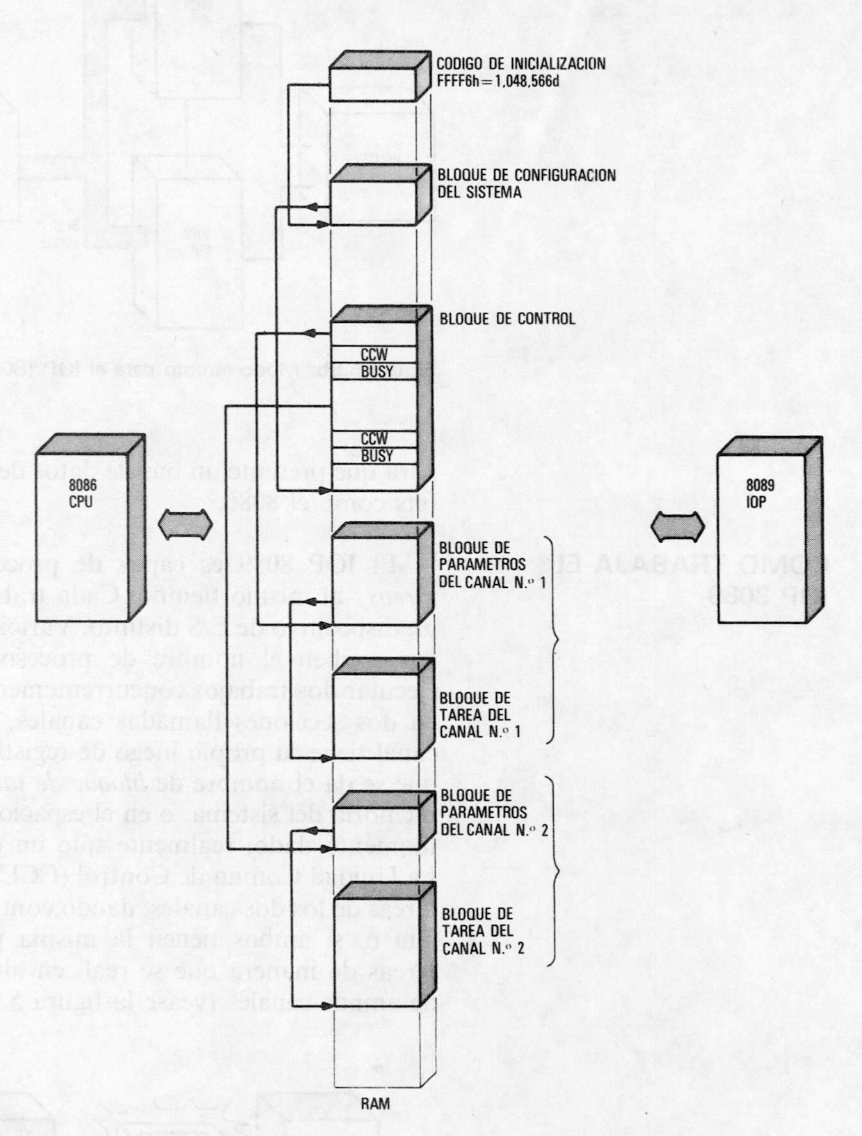


Figura 5.10: Bloques de control, parámetros y tareas para el IOP 8089.

Cuando el IOP 8089 se «borra», comienza cogiendo los 6 octetos superiores de información a partir de una dirección especial de la memoria central (la FFFF6h = 1048566 en decimal). Esta posición está muy cercana a la parte superior de la memoria (FFFFFh)inmediatamente encima de los octetos que la CPU 8086/8088 tiene asignados como punto de comienzo (FFFF0h). A partir de estos bits se calcula la dirección de la siguiente tabla, el

bloque de configuración del sistema. Esta tabla conduce a otra, el bloque de control, que a su vez conduce a otras dos tablas; los bloques de parámetros, uno para cada canal. Cada bloque de parámetros contiene la dirección del bloque de tarea, el programa de gestión de E/S de cada canal. Una vez se ha inicializado el IOP 8089, el bloque de configuración del sistema no vuelve a ser necesario, y puede utilizarse para inicializar otros IOP; pero los bloques de control de parámetros y de tarea deben conservarse para que ambos canales funcionen correctamente. Tanto el procesador central como el IOP 8089 leen y escriben información en los bloques de parámetros y control. Incluso es posible que un IOP colabore en la carga de programas y datos en bloques de tareas y parámetros de otros IOP. Hay unos octetos especiales, uno para cada canal, en el bloque de control llamados octetos de ocupación. Estos octetos se comprueban cada vez que un procesador quiere acceder a un bloque del canal, dado que es fundamental que la CPU y un IOP no accedan nunca al mismo tiempo a un mismo bloque. Podría resultar desastroso que el IOP comenzase a ejecutar una tarea y que, antes de acabar, fuese interrumpido por la CPU y se comenzaran a cargar parámetros y/o programas de una nueva tarea en los bloques de la antigua, no terminada (véase la figura 5.11).

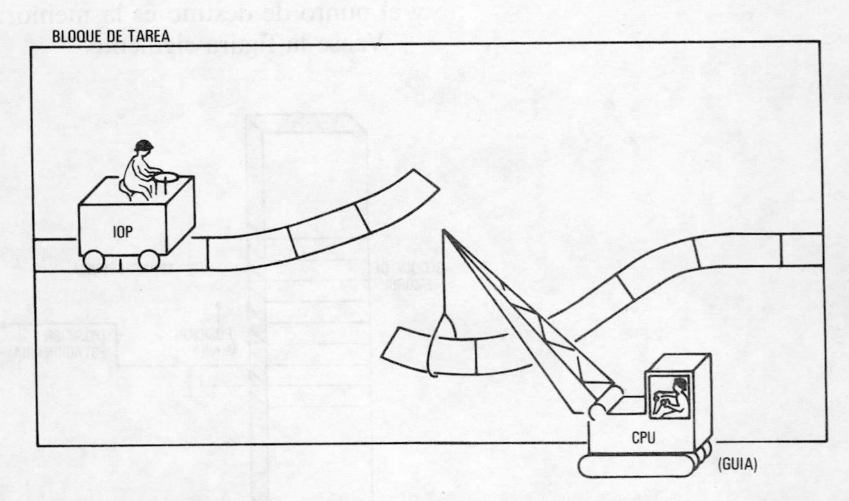


Figura 5.11: IOP y CPU entrando en conflicto.

El IOP 8089 está especialmente diseñado para realizar transferencias a gran velocidad. La fuente y el punto de destino puede ser un controlador de dispositivos, la memoria de E/S, o la memoria central. Este tipo de transferencia rápida recibe el nombre de «acceso directo a memoria» (DMA: Direct Memory Access). La sección que trata del Controlador Programable de DMA 8237 presenta una discursión detallada del sistema DMA. El DMA puede utilizarse para conseguir movimientos de datos rápidos y eficientes entre dispositivos como las unidades de discos

flexibles o computadoras externas a la RAM del sistema, mientras la CPU continúa trabajando en otras cosas que no requieran el bus del sistema.

Podemos imaginar que todo canal del IOP puede actuar de dos modos; regular o DMA. El modo regular resulta útil para cargar parámetros en el IOP y su bloque de parámetros, y para inicializar los chips controladores en vistas a una transferencia. Tras esto, hay una instrucción especial (XFER) que lleva al canal al modo de DMA. En este modo, un registro apunta a la fuente y otro al punto de destino (los registros de IOP 8089 se explican en la sección siguiente). Si la fuente y el punto de destino son posiciones de memoria, después de la transferencia de cada octeto de la palabra se aumenta automáticamente el valor de sus punteros para que apunte a la dirección de memoria siguiente (en la memoria del sistema o en el espacio de E/S), dejándolos preparados para la siguiente transferencia. Si la fuente o el punto de destino es un port de E/S, los punteros no se modifican. Ciertos bits de un registro especial del IOP, el registro del control del canal, determinan qué opciones se toman para la fuente y el destino. Por ejemplo, si se desea hacer una transferencia entre un port de entrada a un bloque de memoria, durante la inicialización hay que dar a los bits de registro de control del canal los valores que determinen que la fuente es un port (no hay que incrementar) y el punto de destino es la memoria (incrementar).

Véase la figura siguiente.

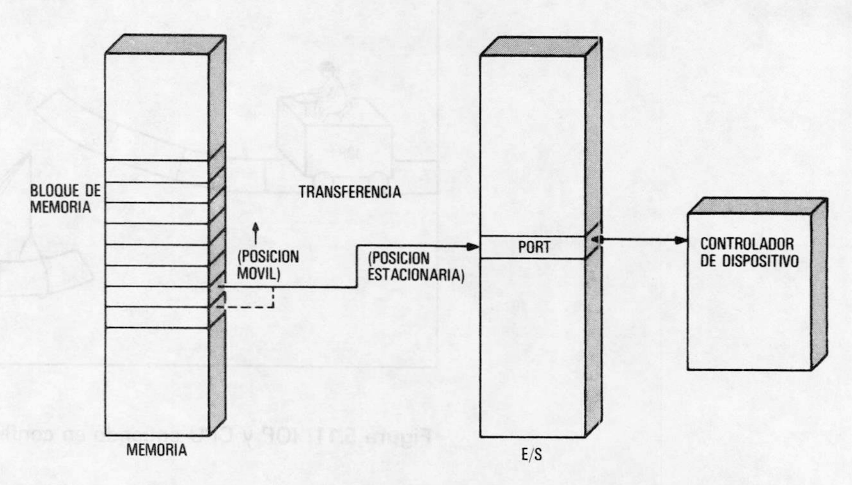


Figura 5.12: La memoria en comparación con las transferencias de E/S.

Hay varias formas de terminar una transferencia del IOP. Y de nuevo, ciertos bits del registro de control del canal son los encargados de determinar cómo se va a hacer. La transmisión puede acabar después de una transferencia simple, según un octeto contador específico, con una comparación enmascarada, o por una señal externa. Las tres últimas posibilidades pueden activarse o desactivarse independientemente. La transferencia

acabará cuando ocurra por primera vez una de las condiciones seleccionadas.

Durante una transferencia puede realizarse a la vez una traducción. Esto es, cada octeto que entra sirve como índice de una tabla de posibles octetos de salida, llamada la tabla de traducción. Si dicha tabla contiene los octetos a(0), a(1), a(2),..., etcétera, un octeto de entrada de valor i produce la salida a(i). Se utiliza uno de los registros del IOP como puntero a dicha tabla (que la ha cargado previamente la CPU o el propio IOP), y uno de los bits del registro de control del canal sirve para activar o desactivar esta opción. La posibilidad de traducción simultánea tiene muchas aplicaciones, como puede ser la traducción de un código ASCII a otro EBCDIC, o la gestión de un esquema de color-intensidad para tratamiento de gráficos. El 8086/8088 posee una instrucción que hace esta traducción automáticamente (capítulo 3), de una forma muy eficiente.

JUEGO DE REGISTROS DEL IOP 8089

La figura 5.13 muestra el juego de registros del IOP 8089. Cada canal de E/S posee 8 registros, 4 de 21 bits y de 4 de 16 bits. Los registros de 21 bits reciben los nombres de registros de propósito general A (GA), B (GB), C (GC) y puntero de tarea

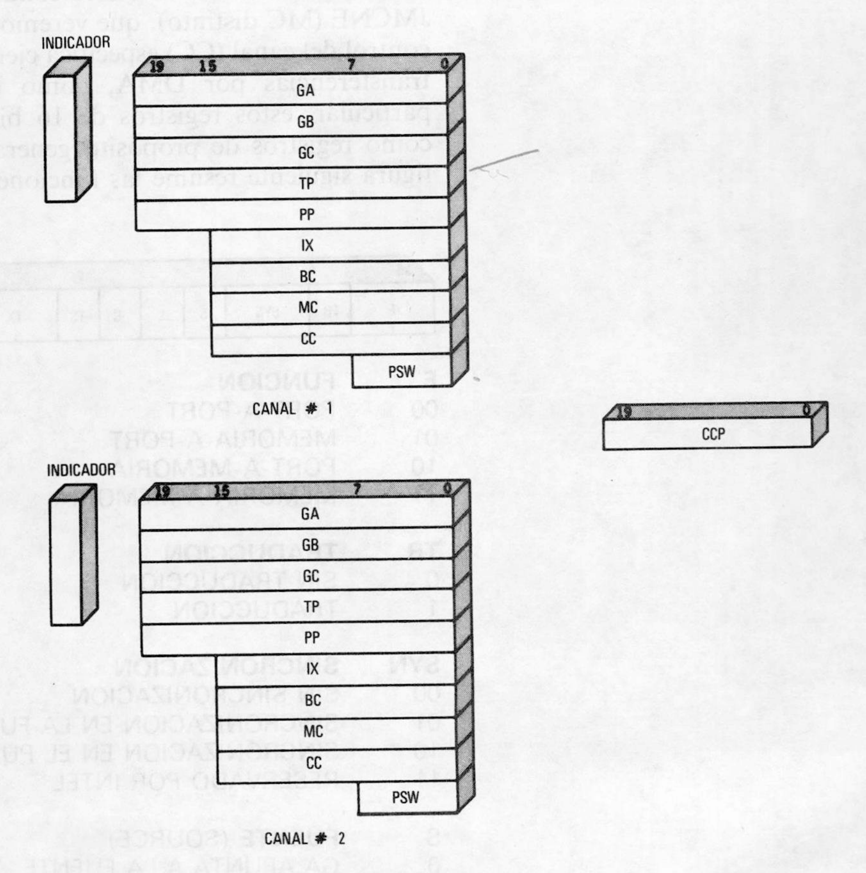


Figura 513: Juego de registros del IOP 8089.

(TP). Su misión va a ser la de contener direcciones de 20 bits, dando así un espacio de memoria direccionable de 1 megaocteto del 8086/8088. El bit número 20 (el más significativo) recibe el nombre de bit indicador y determina según su valor si la dirección se refiere a la memoria principal o al espacio de E/S. Cuando el IOP 8089 direcciona a la memoria central utiliza al completo los 20 bits de dirección. Cuando direcciona al espacio de E/S utiliza sólo 16 bits.

Durante las transferencias por DMA, los registros GA y GB se utilizan alternativamente como punteros a las posiciones de la fuente y destino, y el registro GC apunta a una tabla de traducción o de búsqueda como la mencionada más arriba. El puntero de tarea (TP) es el contador de programa del canal, y apunta en cada momento a la siguiente instrucción del programa. Los registros de 16 bits reciben los nombres de registro índice (IX), contador de octetos (BC), máscara/comparación (MC) y control del canal (CC). El registro índice se utiliza en los dos modos de direccionamiento del IOP que permiten la indexación. Los registros contador de octetos y de máscara/comparación son útiles a la hora de determinar las condiciones de terminación de las transferencias por DMA. El registro MC se utiliza también en las instrucciones JMCE (salto condicional a que MC sea igual) y JMCNE (MC distinto), que veremos más adelante. El registro de control del canal (CC) especifica ciertos parámetros relativos a las transferencias por DMA, como muestra la figura 5.14. En particular, estos registros de 16 bits pueden utilizarse también como registros de propósito general junto a los de 21 bits. La figura siguiente resume las funciones de estos registros.

15 7 0

F	TR	SYN	S	L	C	TS	TX	TBC	TMC

F **FUNCION**
00 PORT-A-PORT
01 MEMORIA-A-PORT
10 PORT-A-MEMORIA
11 MEMORIA-A-MEMORIA

TR **TRADUCCION**
0 SIN TRADUCCION
1 TRADUCCION

SYN **SINCRONIZACION**
00 SIN SINCRONIZACION
01 SINCRONIZACION EN LA FUENTE
10 SINCRONIZACION EN EL PUNTO DE DESTINO
11 RESERVADO POR INTEL

S **FUENTE** (SOURCE)
0 GA APUNTA A LA FUENTE
1 GB APUNTA A LA FUENTE

L	**BLOQUEO** (LOCK)
0	NO HAY BLOQUEO
1	BLOQUEO DURANTE LA TRANSFERENCIA
C	**ENCADENAMIENTO**
0	SIN ENCADENAMIENTO
1	ENCADENADO: PONER TB A PRIORIDAD 1
TS	**FIN DE TRANSFERENCIA SIMPLE**
0	NO HAY TERMINACION DE TRANSFERENCIA
1	FIN DESPUES DE TRANSFERENCIA SIMPLE
TX	**FIN POR SEÑAL EXTERNA**
00	NO HAY TERMINACION EXTERNA
01	FINAL POR SEÑAL EXTERNA ACTIVA; DESPLAZAMIENTO=0
10	FINAL POR SEÑAL EXTERNA ACTIVA; DESPLAZAMIENTO=4
11	FINAL POR SEÑAL EXTERNA ACTIVA; DESPLAZAMIENTO=8
TBC	**FIN POR CONTADOR DE BYTES**
00	NO HAY TERMINACION POR CONTADOR DE BYTES
01	FIN CUANDO BC=0; DESPLAZAMIENTO=0
10	FIN CUANDO BC=0; DESPLAZAMIENTO=4
11	FIN CUANDO BC=0; DESPLAZAMIENTO=8
TMC	**FIN POR MASCARA/COMPARACION**
000	NO HAY TERMINACION POR MC
001	FIN POR IGUAL; DESPLAZAMIENTO=0
010	FIN POR IGUAL; DESPLAZAMIENTO=4
011	FIN POR IGUAL; DESPLAZAMIENTO=8
100	(SIN EFECTO)
101	FIN POR DISTINTO; DESPLAZAMIENTO=0
110	FIN POR DISTINTO; DESPLAZAMIENTO=4
111	FIN POR DISTINTO; DESPLAZAMIENTO=8

Figura 5.14: Registro de control del canal.

Registro	**Tamaño**	**Apunta a**	**Utilizado por el programa como**	**Utilizado por DMA como**
GA	21	memoria o E/S	general y base	puntero de fuente/destino
GB	21	memoria o E/S	general y base	puntero de fuente/destino
GC	21	memoria o E/S	general y base	puntero a la tabla de traducción
TP	21	memoria o E/S	contador de programa	puntero a la tabla de saltos
PP	20	memoria	base	N/A
IX	16	N/A	general e índice	N/A
BC	16	N/A	general	contador de octetos
MC	16	N/A	general y máscara/comparación	máscara/comparación
CC	16	N/A	no se usa	guarda parámetros

Tabla 5.1: Resumen de las funciones de los registros del canal.

Además de los registros de propósito general y de los de DMA, cada canal tiene un puntero de parámetros (PP) y una palabra de estado del programa (PSW: Program Status Word). El registro PP tiene 20 bits y contiene la dirección del bloque de parámetros del canal. La PSW tiene 8 bits y guarda información sobre ciertas condiciones. Ninguno de estos registros es accesible al programa del canal, pero se salvaguardan en memoria cuando le llega al IOP una orden de suspender su actividad y se restauran cuando ésta se reinicia. La CPU puede preguntar y modificar estas posiciones a efectos del control del IOP.

Hay otro registro, el puntero de control de canal (CCP: Channel Control Pointer), que contiene la dirección del bloque de control del canal. Puesto que hay un solo bloque de control de canal por IOP, el IOP 8089 contiene un sólo registro CCP.

DIRECCIONAMIENTO DEL IOP 8089

El IOP 8089 admite cinco modos básicos de direccionamiento: a registro, por registro base, por desplazamiento, indexado, e indexado auto-incremental. En el direccionamiento «a registro», el registro contiene directamente el operando. En el de registro base, uno de los registros GA, GB, GC o PP actúa a modo de puntero guardando la dirección del operando. El tercer modo, por desplazamiento, la dirección del operando se halla sumando un número al contenido de un registro base. En el direccionamiento indexado, la dirección del operando se obtiene sumando

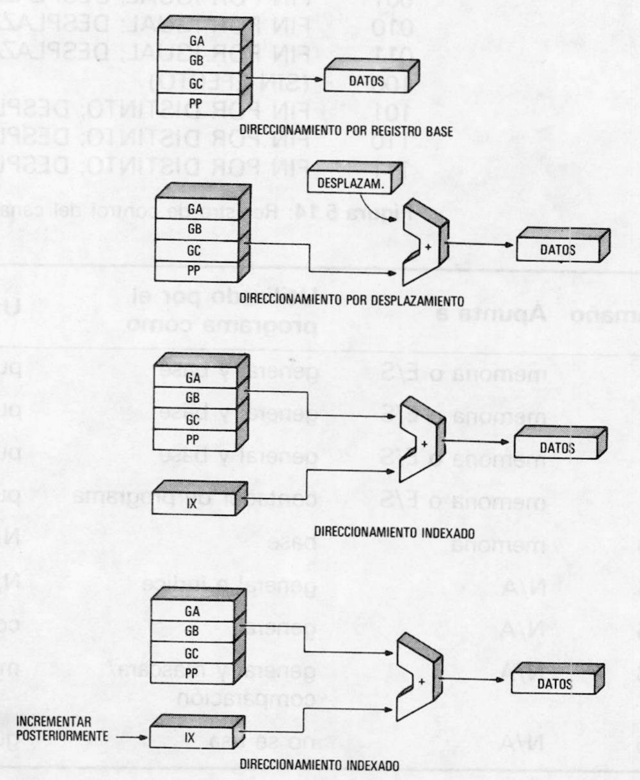


Figura 5.15: Modos de direccionamiento del IOP 8089.

los contenidos de un registro base y el registro índice (IX); y finalmente, en el modo indexado auto-incremental, la dirección del operando se determina de la misma manera que en el indexado, pero además el registro índice se aumenta automáticamente en un cierto valor (1 ó 2 dependiendo del modo de octeto o palabra).

JUEGO DE INSTRUCCIONES DEL IOP

El juego de instrucciones del IOP incluye operaciones de transferencia de datos, aritméticas y lógicas, bifurcaciones condicionales e incondicionales, llamadas a subrutinas, manipulación de bits y verificación, una instrucción especial de transferencia DMA e instrucciones de control del programa. La tabla siguiente resume el juego de instrucciones del IOP 8089.

NEMOTECNICO		DESCRIPCION
MOV	destino, fuente	Transferencia memoria-a-memoria o registro-a-memoria o memoria-a-registro
MOVI	destino, dato	Transfiere el dato (inmediato) a registro o memoria.
MOVP	destino, fuente	Transfiere el puntero de memoria-a-registro o de registro-a-memoria
LPD	registro, memoria	Carga el registro con la dirección de memoria
LPDI	registro, dato	Carga el registro con la dirección del dato (inmediato)
ADD	destino, fuente	Suma registro-a-memoria o memoria-a-registro
ADDI	destino, dato	Suma el dato inmediato a registro o memoria
AND	destino, fuente	Producto lógico de registro-a-memoria o memoria-a-registro
ANDI	destino, dato	Producto lógico del dato inmediato con memoria o registro
OR	destino, fuente	Suma lógica registro-a-memoria o memoria-a-registro
ORI	destino, dato	Suma lógica del dato inmediato con memoria o registro
NOT	destino	Negación lógica del contenido del registro o memoria
DEC	destino	Decrementar el contenido del *registro o memoria*
INC	destino	Incrementar el contenido del registro o memoria

JMP	etiqueta	Salto incondicional a la etiqueta
JMCE	fuente, etiqueta	Salto si máscara/comparación es igual
JMCNE	fuente, etiqueta	Salto si máscara/comparación es distinto
JZ	fuente, etiqueta	Salto si la fuente es cero
JNZ	fuente, etiqueta	Salto si la fuente no es cero
JBT	fuente, bit, etiqueta	Salto si el bit está a 1
JNBT	fuente, bit, etiqueta	Salto si el bit está a 0
CALL	dirección de salvaguarda, etiqueta	Llamada a la subrutina que indica la etiqueta
CLR	destino, bit	Borrar el bit seleccionado
SET	destino, bit	Poner a 1 el bit seleccionado
TSL	destino, valor, etiqueta	Verificar y poner a 1 mientras está bloqueado
NOP		No operación
SINTR		Poner a 1 el bit de servicio de interrupción
WID	anchura de fuente, destino anchura	Determinar anchura de buses
XFER		Entrar en modo DMA

Tabla 5.2: Instrucciones del IOP 8089.

Hay varias instrucciones que merecen especial atención.

Las introducciones JMCE y JMCNE utilizan el registro máscara/comparación (MC) del IOP. Ambas instrucciones tienen una fuente y una etiqueta. Se compara el registro MC con la fuente y dependiendo de la igualdad o no de éstos, se salta a la instrucción indicada en la etiqueta. La comparación se realiza de la siguiente manera:

Para cada bit del octeto de máscara que sea 1, los bits correspondientes del octeto fuente y del octeto de comparación deben coincidir.

Esto genera una operación general y potente de comparación. Sin necesidad de modificar datos ni ningún registro, se puede detectar cuándo un cierto conjunto de bits siguen un esquema particular. Por ejemplo, supongamos que queremos realizar una bifurcación cuando un cierto octeto de datos es una A en código ASCII, ya sea mayúscula o minúscula, y sin que importe si el bit más significativo (el bit número 7, el de control de paridad) está a 1 o a 0. Puesto que el código para la A es 41H=01000001B, y para la *a* es 61H=01100001B, el bit número 5 puede ser 0 ó 1 indistintamente, y por tanto no se incluirá en la máscara. Si tampoco queremos incluir el bit de paridad, la máscara será 5FH=01011111B (todo unos salvo los bits 5 y 7). Sólo se examinarán los bits que son 1 en la máscara, y se ignorarán los

que son 0 (el 5 y el 7). El octeto de comparación puede ser cualquiera de los siguientes: 41H=01000001B; 61H=01100001B; C1H=11000001B, o E1H=11100001B. Cada uno de ellos genera una comparación correcta. (Véase la figura 5.16.)

La instrucción XFER es también interesante. Esta instrucción hace que el canal entre en modo DMA. En este modo, el canal transfiere, automática y rápidamente, un bloque de datos de una posición a otra. Estas posiciones pueden ser de memoria, o ports de E/S, y pueden estar colocadas en la memoria central o en el espacio de E/S. Cada una de estas opciones puede determinarse independientemente para la fuente y el destino. Otra característica interesante de este tipo de transferencias es la posibilidad de pasar automáticamente de buses de 8 bits a 16 bits y viceversa. Tanto la fuente como el destino pueden tener una anchura de 8 o 16 bits (cada uno de ellos por separado). La instrucción WID se encarga de determinar las anchuras de los buses.

La dirección de la fuente se encuentra en uno de los registros GA o GB, y la dirección del punto de destino en el otro. Uno de los bits del registro de control del canal determina exactamente en cuál. Como ya mencionamos antes los bits del registro de control definen también la condición de final de transferencia. La transmisión puede acabar, o bien después de una transferencia simple, o de tres maneras distintas: según un contador de octetos específico (en el registro BC), por una comparación enmascarada (registro MC), o mediante una señal externa (a través del terminal EXT del canal). Cada una de estas tres maneras puede activarse o desactivarse independientemente. La transferencia acaba cuando una de las tres condiciones se cumple.

La traducción (como ya vimos) puede realizarse simultáneamente a la transmisión. Otra vez, un bit de registro de control del canal se encarga de activar o desactivar la opción. El registro GC apunta el comienzo de la tabla de traducción.

La velocidad de ejecución de las instrucciones de la transferencia depende de los tamaños de los buses fuente y destino; de si se realiza traducción simultánea o no, y de otros factores como la

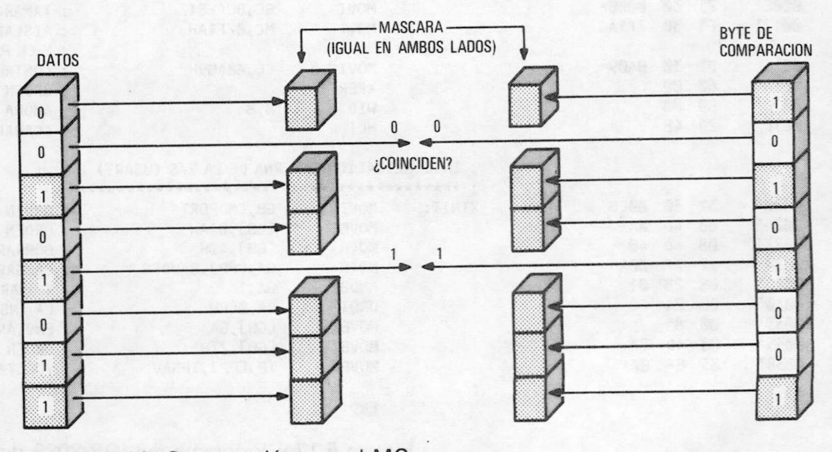


Figura 5.16: Comparación con el MC.

forma de sincronización (que no se discutirá aquí). Son necesarios un mínimo de ocho ciclos de reloj por transferencia. Este mejor caso se da cuando los buses fuente y destino tienen el mismo tamaño y no hay traducción. Si hay traducción se necesitan siete ciclos de reloj adicionales, y si las anchuras de los buses fuente y destino son distintos, el número de ciclos aumenta en 4 a 8. En general, una transferencia le toma al IOP 8089 mucho menos tiempo que a la CPU, pero más tiempo que a un controlador de DMA menos inteligente.

Parece con esto que cuanto más inteligente es el dispositivo, más lento opera. Esto se debe a que la inteligencia del dispositivo está directamente relacionada con el número y complejidad de las decisiones que debe tomar para seleccionar una entre varias opciones.

Cada decisión requiere una buena cantidad de trabajo (tiempo). En otras palabras, un dispositivo menos inteligente no tiene que decidir lo que debe hacer en cada momento, sencillamente hace aquella tarea (única) para la que se diseñó. Por otro lado, existen técnicas que aumentan la eficiencia de los dispositivos, como el pipeline, procesos-paralelos y partes operativas mayores (véase el capítulo 2). Conforme los dispositivos se van haciendo más inteligentes, van siendo más necesarias estas técnicas para mantener y mejorar la tasa de funcionamiento respecto a los dispositivos anteriores, menos inteligentes.

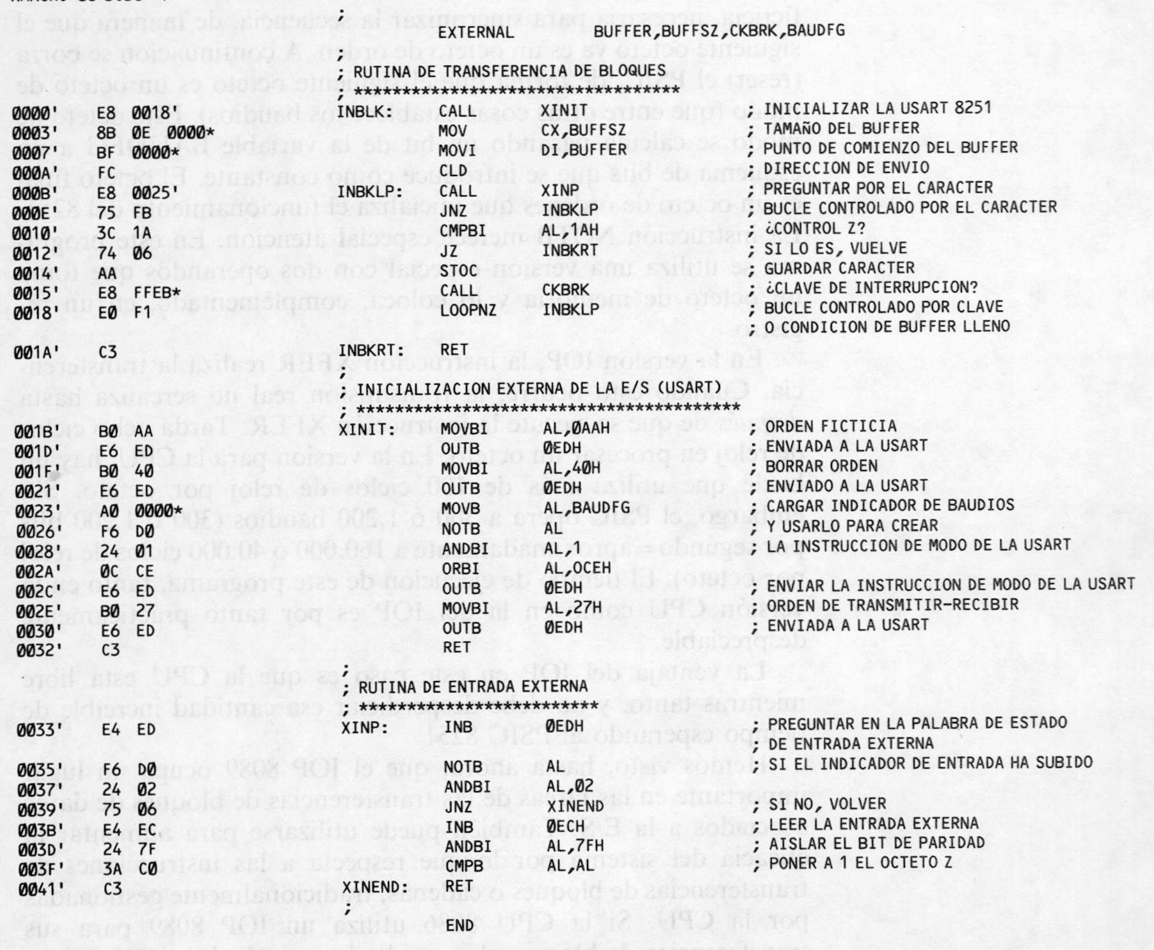
PROGRAMAS-EJEMPLO DEL IOP 8089

La figura 5.17 muestra un programa-ejemplo corto, que utiliza el IOP 8089 para inicializar el Controlador Programable Serie de Interfaz 8155 (PSIC) y entra un bloque de datos desde él. La

```
                           IOPORT    EQU       OECH
                           CMDPORT   EQU       OEDH
                                     EXTERNAL BUFFER,BUFFSZ,TPSAVE, BAUDFG
                           ;
                           ; RUTINA DE TRANSFERENCIA DE BLOQUES
                           ;*********************************
0000'  8B 9F 00* 1C        BEGIN:    CALL      [PP],TPSAV,XINIT    ; INICIALIZAR USART
0004'  11 30 00EC                    MOVI      GA,IOPORT           ; PORT DE DATOS
0008'  31 08 00000* 0000*            LPDI      GB,BUFFER           ; COMIENZO DEL BUFFER (ALMACENAMIENTO TEMPORAL)
000E'  71 30 0000*                   MOVI      BC,BUFFSIZ          ; TAMAÑO DEL BUFFER
0012'  F1 30 7F1A                    MOVI      MC,07F1AH           ; AISLAR PARIDAD
                                                                   ; Y CK PARA CONTROL Z
0016'  D1 30 BA09                    MOVI      CC,08A09H           ; CONTROLES PARA XFER
001A'  60 00                         XFER                          ; ORDEN DE TRANSFERIR
001C'  80 00                         WID       8,8                 ; AHORA LO HACE
001E'  20 48                         HLT                           ; ACABADO, ASI QUE PARA

                           ;
                           ; INICIALIZACION EXTERNA DE LA E/S (USART)
                           ; ***************************************
0020'  31 30 00ED          XINIT:    MOVI      GB,CMDPORT          ; ORDEN DIRECCION DEL PORT DE LA USART
0024'  08 4D AA                      MOVBI     [GB],0AAH           ; ORDEN FICTICIA DE LA USART
0027'  08 4D 40                      MOVBI     [GB],40H            ; BORRAR ORDEN
002A'  02 AF 00*                     NOTB      GA,[PP],BAUDFG      ; CARGAR INDICADOR DE BAUDIOS
002D'  08 28 01                      ANDBI     GA,1                ; Y USARLO PARA CREAR
0030'  08 24 CE                      ORBI      GA,0CEH             ; LA INSTRUCCION DE MODO DE LA USART
0033'  00 85                         MOVB      [GB],GA             ; ENVIAR LA INSTRUCCION DE MODO DE LA USART
0035'  08 4D 27                      MOVBI     [GB],27H            ; ORDEN DE TRANSMITIR/RECIBIR
0038'  83 8F 00*                     MOVP      TP,[PP],TPSAV       ; VUELTA
                           ;
                                     END
```

Figura 5.17a: Programa del IOP 8089 de inicialización y uso del 8251.



```
XMACRO-86 3.36  1
                          ;
                             EXTERNAL       BUFFER,BUFFSZ,CKBRK,BAUDFG
                          ;
                          ; RUTINA DE TRANSFERENCIA DE BLOQUES
                          ; **********************************
0000'  E8 0018'   INBLK:     CALL       XINIT          ; INICIALIZAR LA USART 8251
0003'  8B 0E 0000*           MOV        CX,BUFFSZ      ; TAMAÑO DEL BUFFER
0007'  BF 0000*              MOVI       DI,BUFFER      ; PUNTO DE COMIENZO DEL BUFFER
000A'  FC                    CLD                       ; DIRECCION DE ENVIO
000B'  E8 0025'   INBKLP:    CALL       XINP           ; PREGUNTAR POR EL CARACTER
000E'  75 FB                 JNZ        INBKLP         ; BUCLE CONTROLADO POR EL CARACTER
0010'  3C 1A                 CMPBI      AL,1AH         ; ¿CONTROL Z?
0012   74 06                 JZ         INBKRT         ; SI LO ES, VUELVE
0014'  AA                    STOC                      ; GUARDAR CARACTER
0015'  E8 FFEB*              CALL       CKBRK          ; ¿CLAVE DE INTERRUPCION?
0018'  E0 F1                 LOOPNZ     INBKLP         ; BUCLE CONTROLADO POR CLAVE
                                                       ; O CONDICION DE BUFFER LLENO
001A'  C3         INBKRT:    RET
                          ;
                          ; INICIALIZACION EXTERNA DE LA E/S (USART)
                          ; ****************************************
001B'  B0 AA      XINIT:     MOVBI      AL,0AAH        ; ORDEN FICTICIA
001D'  E6 ED                 OUTB       0EDH           ; ENVIADA A LA USART
001F'  B0 40                 MOVBI      AL,40H         ; BORRAR ORDEN
0021'  E6 ED                 OUTB       0EDH           ; ENVIADO A LA USART
0023'  A0 0000*              MOVB       AL,BAUDFG      ; CARGAR INDICADOR DE BAUDIOS
0026'  F6 D0                 NOTB       AL             ; Y USARLO PARA CREAR
0028'  24 01                 ANDBI      AL,,           ; LA INSTRUCCION DE MODO DE LA USART
002A'  0C CE                 ORBI       AL,OCEH
002C'  E6 ED                 OUTB       0EDH           ; ENVIAR LA INSTRUCCION DE MODO DE LA USART
002E'  B0 27                 MOVBI      AL,27H         ; ORDEN DE TRANSMITIR-RECIBIR
0030'  E6 ED                 OUTB       0EDH           ; ENVIADA A LA USART
0032'  C3                    RET
                          ;
                          ; RUTINA DE ENTRADA EXTERNA
                          ; *************************
0033'  E4 ED      XINP:      INB        0EDH           ; PREGUNTAR EN LA PALABRA DE ESTADO
                                                       ; DE ENTRADA EXTERNA
0035'  F6 D0                 NOTB       AL             ; SI EL INDICADOR DE ENTRADA HA SUBIDO
0037'  24 02                 ANDBI      AL,02
0039'  75 06                 JNZ        XINEND         ; SI NO, VOLVER
003B'  E4 EC                 INB        0ECH           ; LEER LA ENTRADA EXTERNA
003D'  24 7F                 ANDBI      AL,7FH         ; AISLAR EL BIT DE PARIDAD
003F'  3A C0                 CMPB       AL,AL          ; PONER A 1 EL OCTETO Z
0041'  C3         XINEND:    RET
                          ;
                             END
```

Figura 5.17b: Programa equivalente para el 8086.

figura también muestra un programa equivalente para el 8086/8088. La rutina de inicialización establece la velocidad de trasmisión del SIC 8251 a 300 ó 1.200 baudios, dependiendo del parámetro BAUDFG. En la versión para el IOP 8089, dicho parámetro se guarda en el bloque de parámetros y se llega a él tomando el Pointer de Parámetros (PP) como base, y mediante un desplazamiento. El PP se carga durante la inicialización del IOP, cuando se apunta al comienzo del bloque de parámetros. El IOP suma el desplazamiento a PP para obtener la dirección de BAUDFG dentro del bloque de parámetros.

El desplazamiento no se especifica durante el ensamblaje en nuestro programa, sino que su valor se determina automáticamente mediante *un programa montador*.

Nótese que se envían 4 octetos al port del PSIC 8251. El PSIC espera un octeto de modo y después una serie de octetos de

órdenes. El primer octeto que se envía (AAH) es una orden ficticia, necesaria para sincronizar la secuencia, de manera que el siguiente octeto ya es un octeto de orden. A continuación se borra (reset) el PSIC, de forma que el siguiente octeto es un octeto de modo (que entre otras cosas establece los baudios). Este octeto de modo se calcula pasando un bit de la variable BAUDFG a un esquema de bits que se introduce como constante. El octeto final es un octeto de órdenes que inicializa el funcionamiento del 8251. La instrucción NOTB merece especial atención. En este programa se utiliza una versión especial con dos operandos que toma un octeto de memoria y lo coloca, complementado, en un registro.

En la versión IOP, la instrucción XFER realiza la transferencia. Cuando esto ocurre, la transmisión real no serealiza hasta *después* de que se ejecute la instrucción XFER. Tarda ocho ciclos de reloj en procesar un octeto. En la versión para la CPU, hay un bucle que utiliza más de 100 ciclos de reloj por octeto. Sin embargo, el PSIC opera a 300 ó 1.200 baudios (300 ó 1.200 bits por segundo=aproximadamente a 160.000 ó 40.000 ciclos de reloj por octeto). El tiempo de ejecución de este programa, tanto en la versión CPU como en la del IOP es por tanto prácticamente despreciable.

La ventaja del IOP en este caso es que la CPU está libre mientras tanto, y no debe desperdiciar esa cantidad increíble de tiempo esperando al PSIC 8251.

Hemos visto, hasta ahora, que el IOP 8089 ocupa un lugar importante en las tareas de las transferencias de bloques de datos asociados a la E/S. También puede utilizarse para aumentar la eficacia del sistema por lo que respecta a las instrucciones de transferencias de bloques o cadenas, tradicionalmente gestionadas por la CPU. Si la CPU 8086 utiliza un IOP 8089 para sus transferencias de bloques, los resultados excederán posiblemente a los del Z8000 o el MC68000, sobre todo si se necesita algún control especial de terminación de transferencia.

La figura 5.18 muestra un programa que utiliza el IOP 8089 en la transferencia de bloques de memoria a memoria. La dirección inicial del bloque fuente es SOURCE, la del bloque de destino de DEST y el tamaño máximo de bloque (en octetos) se guarda en la variable COUNT. Las tres cantidades se almacenan en el bloque de parámetros. Si un carácter cualquier (con el bit de paridad aislado) es una vuelta de carro, la transferencia acaba. La figura también muestra un programa similar escrito para el 8086, sin IOP. El programa IOP transfiere información a una velocidad máxima de ocho ciclos de reloj por octeto (si tiene el bus para él mismo). Con un reloj de 5 MHz, esta velocidad representa el poder mover 64 K de memoria en algo así como una décima de segundo. Al no poder hacer la CPU lo que el IOP hace en una única instalación, la CPU necesita entrar en un bucle que se ejecuta en aproximadamente 45 ciclos de reloj. Para esta tarea particular, la CPU necesita un tiempo de seis veces mayor que el IOP.

```
0000'   03 8B 00*        BEGIN:   LPD      GA,[PP],SOURCE    ; DIRECCION DE LA FUENTE
0003'   23 8B 00*                 LPD      GB,[PP],DEST      ; DIRECCION DEL DESTINO
0006'   63 83 00*                 MOV      BC,[PP],COUNT     ; TAMAÑO MAXIMO DEL BLOQUE
0009'   F1 30 7F0D                MOVI     MC,07F0DH         ; AISLAR PARIDAD
                                                             ; Y CK PARA LA VUELTA DE CARRO
000D'   D1 30 C209                MOVI     CC,0C209H         ; CONTROL PARA LA TRANSFERENCIA
0011'   60 00                     XFER                       ; LE DICE QUE DEBE TRANSFERIR
0013'   80 00                     WID      8,8               ; AQUI TRANSFIERE
0015'   20 48                     HLT                        ; ACABADO, ASI SE PARA
                         ;
                                  END
```

Figura 5.18a: Programa de transferencia memoria-a-memoria utilizando el IOP 8089.

```
      XMACRO-86 3,36   1
                         ;
                                  EXTERNAL      SOURCE,DEST,COUNT

                         ;
                         ; RUTINA DE TRANSFERENCIA DE BLOQUES
                         ; **********************************
0000'   8B 36 0000*      BEGIN:   MOV      SI,SOURCE         ; DIRECCION DE LA FUENTE
0004'   8B 3E 0000*               MOV      DI,DEST           ; DIRECCION DE DESTINO
0008'   8B 0E 0000*               MOV      CX,COUNT          ; COUNT
000C'   FC                        CLD                        ; DIRECCION DE ENVIO
000D'   AC               LOOP:    LODC                       ; OBTENER EL BYTE
000E'   8A E0                     MOVB     AH,AL             ; SALVAGUARDARLO
0010'   24 7F                     ANDBI    AL,7FH            ; AISLAR PARIDAD
0012'   3C 0D                     CMPBI    AL,0DH            ; PREGUNTAR POR EL CARACTER DE VUELTA DE CARRO
0014'   8A C4                     MOVB     AL,AH             ; RESTAURARLO
0016'   AA                        STOC                       ; GUARDAR EL BYTE
0017'   E0 F4                     LOOPNZ   LOOP              ; BUCLAR HASTA ACABAR
0019'   C3                        RET
                         ;
                                  END
```

Figura 5.18b: Programa equivalente para el 8086.

CONCLUSIONES

En este capítulo hemos visto un resumen de las características del IOP 8089. Hemos estudiado cómo trabaja, ejecutando sus propios programas en un contexto de multiproceso bajo la dirección de una CPU 8086/8088, y hemos visto cómo puede liberar a la CPU del trabajo que ocasiona la lentitud de los dispositivos de E/S. Hemos visto también cómo el IOP es capaz de transferir bloques de memoria inteligentemente, a velocidades mucho mayores que la CPU.

Hay ciertos comentarios que merecen tenerse en cuenta respecto al IOP 8089. Intel ha mencionado alguna vez su esperanza de que pronto haya especialistas en E/S que programen con este dispositivo, y que prácticamente no necesiten nunca trabajar con la CPU. Si quiere aprender a programar el 8089, cuidado, que lo explicado en este capítulo no es más que una introducción al tema. Nuestro consejo es que para ello lea las secciones correspondientes del *The 8086 Family User's Manual*.

Capítulo 6

Los chips de soporte del 8086/8088

En este capítulo veremos una serie de chips que proveen al 8086/8088 circuitería para tres tipos de funciones: lógica para generación de pulsos de reloj, lógica para la interfaz de bus y controladores. Aunque estos chips no son tan «atractivos» como los chips procesadores vistos en los tres últimos capítulos, realizan funciones vitales para el sistema, como la sincronización, la conexión de los procesadores con el resto del sistema y la conexión del computador con el mundo exterior.

LOGICA DEL GENERADOR DE PULSOS DE RELOJ

Cualquier sistema basado en el 8086/8088 requiere una lógica adicional encargada de generar las señales de sincronización para todo el sistema. El Generador de Pulsos de Reloj 8284 de Intel, junto con un cristal oscilador (cristal) externo, es un chip diseñado específicamente para estas tareas. Es de resaltar que la CPU 8085 lleva incorporada la circuitería del 8284 en el mismo procesador. Se espera que las versiones futuras del 8086, el iAPX 186 y otros, también lo lleven.

El generador de pulsos de reloj 8284

El 8284 (véase la figura 6.1) es un chip con 18 terminales que puede utilizarse para generar económicamente los pulsos de reloj para el 8086, 8088, 8087 y 8089 y sus periféricos. Los pulsos de reloj determinan la velocidad de funcionamiento del sistema. La velocidad máxima estándar para estos procesadores es una frecuencia de reloj de 5 MHz (es decir, $1/(5*10^6)=0{,}2*10^{-6}=200$ nanosegundos por ciclo), aunque algunos chip particulares funcionan a 8 MHz (=125 nanosegundos por ciclo). Tanto los chips estándar, como las versiones especiales más rápidas admiten teóricamente una velocidad mínima de 2 MHz, aunque en la práctica funcionan incluso a frecuencias menores.

El Generador de Pulsos 8284 necesita un cristal oscilador, o una señal lógica externa como fuente de frecuencia. La opción se especifica conectando un terminal o bien a tierra, o bien a la fuente de alimentación de 5 voltios. La fuente de frecuencia proporciona una frecuencia triple de la señal resultante del 8284. Poniendo un cristal oscilador u otro, la frecuencia obtenida puede variar entre 5 MHz hasta casi los 8 MHz. Si se usa una señal externa, la velocidad puede variar desde un ciclo cada vez (control manual) hasta unos 8 MHz.

Para conseguir un rendimiento óptimo de los procesadores, los pulsos de reloj generados por el 8284 se mantienen a *tensión* alta durante un 33 por 100 del período. Como la frecuencia de la fuente es triple que la de salida del 8284, es relativamente fácil

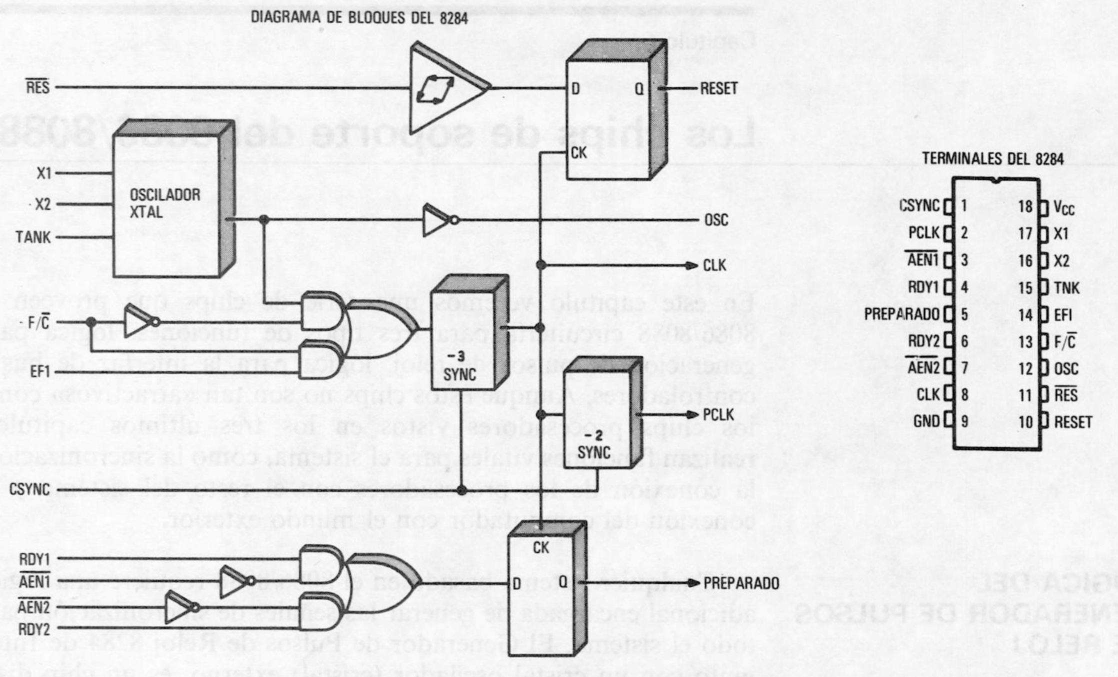


Figura 6.1: Generador de señales de reloj 8284. Terminales y estructura interna.

generar este tipo de onda. Para entender cómo, imaginémonos al músico encargado del tambor en una orquesta, que está tocando una melodía familiar al ritmo de «bump, pa, pa». El es el encargado de hacer el «bump».

Lo que puede ir haciendo es sencillamente ir contando 1, 2, 3 y darle al tambor cuando llegue a 3, volviendo a comenzar el conteo. El Generador de Pulsos de Reloj 8284 hace exactamente lo mismo. El «ritmo básico» de 1, 2, 3 le proporciona la fuente de frecuencia externa y cada tres pasos corresponden a un ciclo completo de reloj. Como el encargado del tambor, el 8284 necesita contar hasta 3 una y otra vez, poniendo la salida a tensión alta mientras el contador es 1 y a cero mientras el contador es 2 ó 3.

Normalmente salen tres señales del Generador de Pulsos de Reloj 8284 hacia el procesador. Estas tres señales son CLK (señal de reloj), RESET y la READY. Las dos últimas señales están sincronizadas con CLK.

La misión de la señal RESET es la de reinicializar los valores de la computadora como si ésta se hubiese apagado y vuelto a enchufar. Esta señal es imprescindible para solventar los casos en que un programa se mete en un bucle infinito, o que un ruido de alimentación afecte a partes del programa en curso. Desenchufar la máquina y volverla a enchufar puede ser peligroso porque se generan sobretensiones que pueden dañar a parte de los componentes eléctricos del circuito. Con la señal RESET se consigue el mismo resultado lógico sin ninguno de estos problemas.

La función de la señal READY es la de sincronizar el procesador con los dispositivos externos más lentos. La señal READY va desde el dispositivo externo al procesador, pasando a través del generador de pulsos de reloj. Cuando el procesador quiere acceder a un dispositivo que no está preparado para la transferencia, el dispositivo envía un 0 por línea READY. Cuando el procesador recibe esta señal, entra en un ciclo de espera hasta que aparece un 1 por dicha línea. Sólo entonces continúa el programa.

El 8284 produce otra señal, PCLK (reloj periférico), que funciona a una frecuencia mitad que CLK, con un ciclo de trabajo al 50 por 100 (un 50 por 100 a nivel de tensión alta y un 50 por 100 a nivel cero). Está previsto para sincronizar aquella lógica que requiera esta forma (más antigua) de temporización.

LOGICA DE INTERFAZ DEL BUS

La lógica de interfaz del bus es necesaria por dos razones: 1) las señales de los procesadores pueden no ser lo suficientemente potentes para controlar al resto del sistema y 2) las señales producidas por los procesadores puede que no correspondan directamente a las señales que necesita el resto del sistema.

El Controlador de Bus 8288, el Transceptor de Datos Octal 8286, y el Latch Octal 8282 (véanse las figuras 6.2-6.4) se utilizan

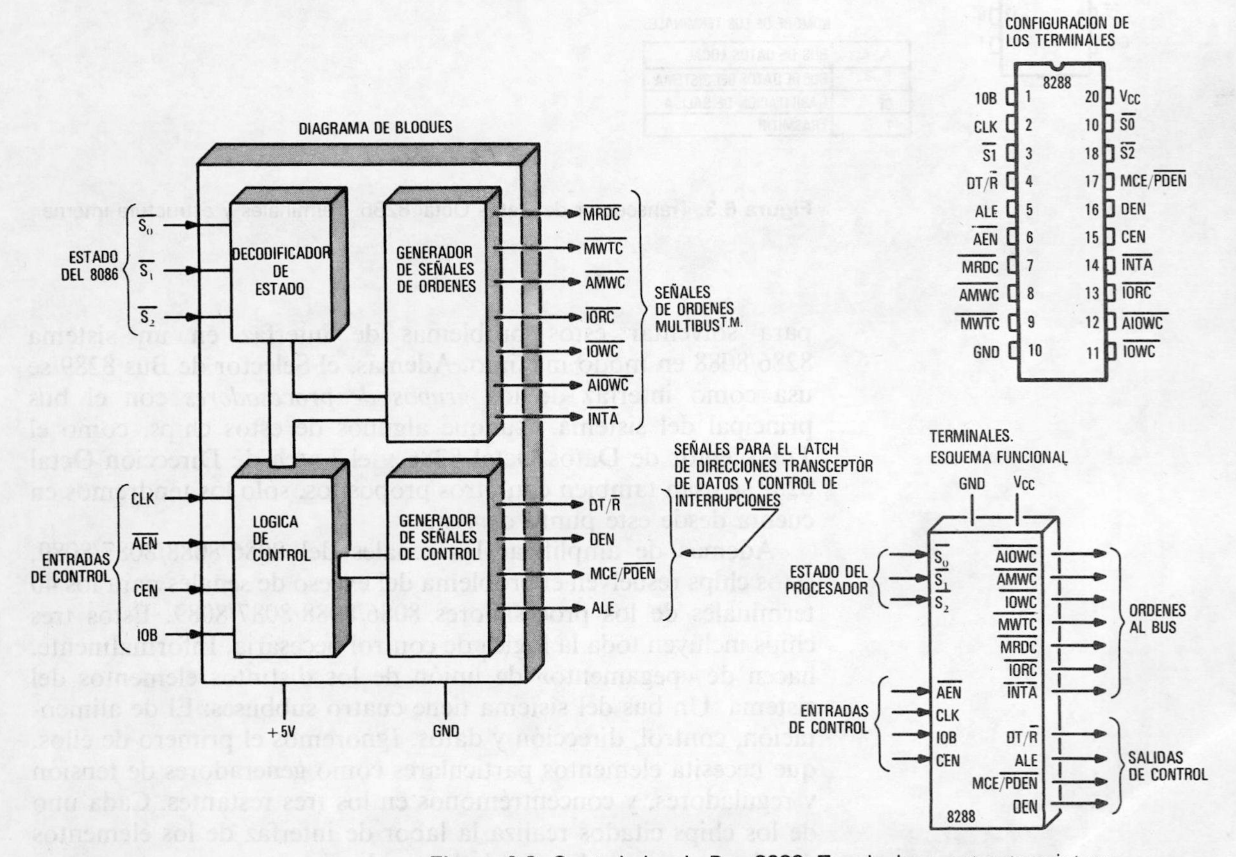
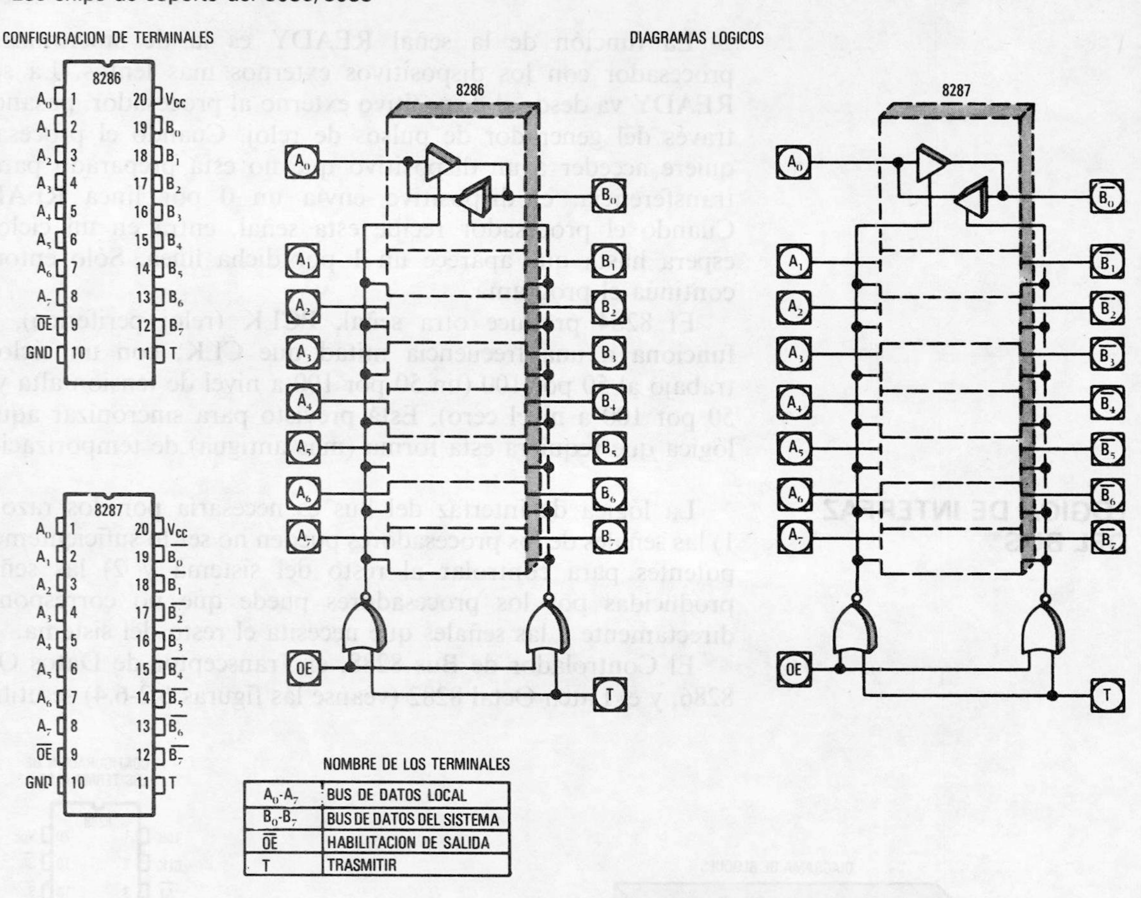


Figura 6.2: Controlador de Bus 8288. Terminales y estructura interna.

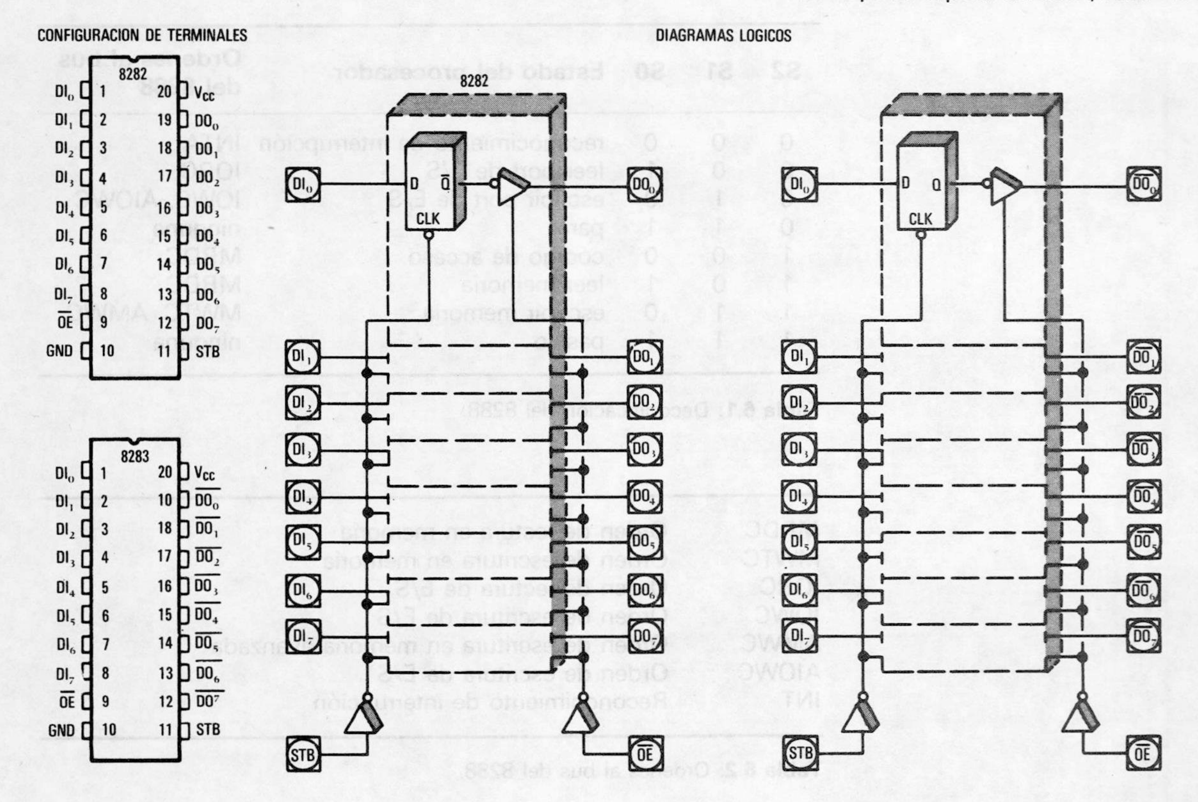


A_0-A_7	BUS DE DATOS LOCAL
B_0-B_7	BUS DE DATOS DEL SISTEMA
$\overline{OE}$	HABILITACION DE SALIDA
T	TRASMITIR

Figura 6.3: Transceptor de Datos Octal 8286. Terminales y estructura interna.

para solventar estos problemas de interfaz en un sistema 8286/8088 en modo máximo. Además, el Selector de Bus 8289 se usa como interfaz de los *grupos de procesadores* con el bus principal del sistema. Aunque algunos de estos chips, como el Transceptor de Datos Octal 8286 y el Latch de Dirección Octal 8282 se usan también con otros propósitos, sólo los tendremos en cuenta desde este punto de vista.

Además de amplificar las señales del 8086/8088/8087/8089, estos chips resuelven el problema del exceso de señales para los 40 terminales de los procesadores 8086/8088/8087/8089. Estos tres chips incluyen toda la lógica de control necesaria. Informalmente, hacen de «pegamento» de unión de los distintos elementos del sistema. Un bus del sistema tiene cuatro subbuses: El de alimentación, control, dirección y datos. Ignoremos el primero de ellos, que necesita elementos particulares como generadores de tensión y reguladores, y concentrémonos en los tres restantes. Cada uno de los chips citados realiza la labor de interfaz de los elementos del sistema con cada uno de los subbuses.



NOMBRES DE LOS TERMINALES

DI_0-DI_7	ENTRADA DE DATOS
DO_0-DO_7	SALIDA DE DATOS
$\overline{OE}$	HABILITACION DE SALIDA
STB	HABILITACION (STROBE)

Figura 6.4: Latch de Direcciones Octal 8282. Terminales y estructura interna.

El Controlador de Bus 8288

El Controlador de Bus 8288 es un chip de 20 terminales, encargado de realizar el interfaz entre el procesador y el bus de control. Decodifica las señales de estado S0, S1 y S2 del 8086/8088 en modo máximo y genera un conjunto completo de señales de control, como la de control de lectura en memoria (MRDC), control de lectura de E/S (IORC), control de escritura en memoria (MWTC), control de escritura de E/S (IOWC), latch de direcciones disponible (ALE) y datos disponibles (DEN). Algunas de estas señales de control, como las de lectura y escritura, tienen como destino el bus del sistema, mientras que otras, tales como las de direcciones o datos disponibles, son señales destinadas a los otros chips de interfase del procesador con los otros subbuses (el Transceptor 8286 y el Latch Octal 8286). Hay también algunas entradas al 8288 de otros dispositivos. Las tablas 6.1 y 6.2 muestran las señales de bus producidas por el Controlador de Bus 8288 en respuesta a las señales de estados de los procesadores.

S2	S1	S0	Estado del procesador	Ordenes al bus del 8288
0	0	0	reconocimiento de interrupción	INTA
0	0	1	leer port de E/S	IORC
0	1	0	escribir port de E/S	IOWC, AIOWC
0	1	1	parar	ninguna
1	0	0	código de acceso	MRDC
1	0	1	leer memoria	MRDC
1	1	0	escribir memoria	MWTC, AMWC
1	1	1	pasivo	ninguna

Tabla 6.1: Decodificación del 8288.

MRDC	Orden de lectura en memoria
MWTC	Orden de escritura en memoria
IORC	Orden de lectura de E/S
IOWC	Orden de escritura de E/S
AMWC	Orden de escritura en memoria avanzada
AIOWC	Orden de escritura de E/S
INT	Reconocimiento de interrupción

Tabla 6.2: Ordenes al bus del 8288.

El Transceptor 8286

El Transceptor 8286 es un chip de 20 terminales cuya misión es servir de interfaz entre el procesador y el bus de datos. Se utiliza como almacenamiento intermedio para los datos que llegan y salen del procesador, y es necesario por varias razones. Por ejemplo, algunas veces las líneas de datos del procesador no pueden suministrar la corriente necesaria para la carga de un dispositivo externo, y tienen que amplificarse antes de llegar al bus del sistema. Otro uso del transceptor es el de convertir las señales bidireccionales que ciertos buses de datos externos necesitan en señales unidireccionales. Una tercera posibilidad es la de ayudar a distinguir las señales de datos sobre las señales de direcciones. Observe que en el 8086/8088, las señales de datos y direcciones están *multiplexadas* en los mismos terminales.

Latch Octal 8282

El Latch (memoria de enclavamiento) Octal 8282 es otro chip también de 20 terminales, encargado de hacer de interfaz entre las líneas multiplexadas de datos/direcciones del 8086/8088 y el bus de direcciones del sistema. El 8282 espera hasta que la información sobre la dirección aparece en tales terminales, y entonces toma la información y la mantiene sobre el bus de direcciones del sistema. Con la línea de latch de dirección disponible (ALE), el procesador (modo mínimo) o el Controlador de Bus 8088 (modo máximo) le dice al Latch Octal cuando debe tomar esta información.

Todos estos chips de interfaz pueden llevarse a un estado de alta independencia (desconectados eléctricamente del bus) cuando otros procesadores o controladores poseen el control del bus. Véase la figura 6.5.

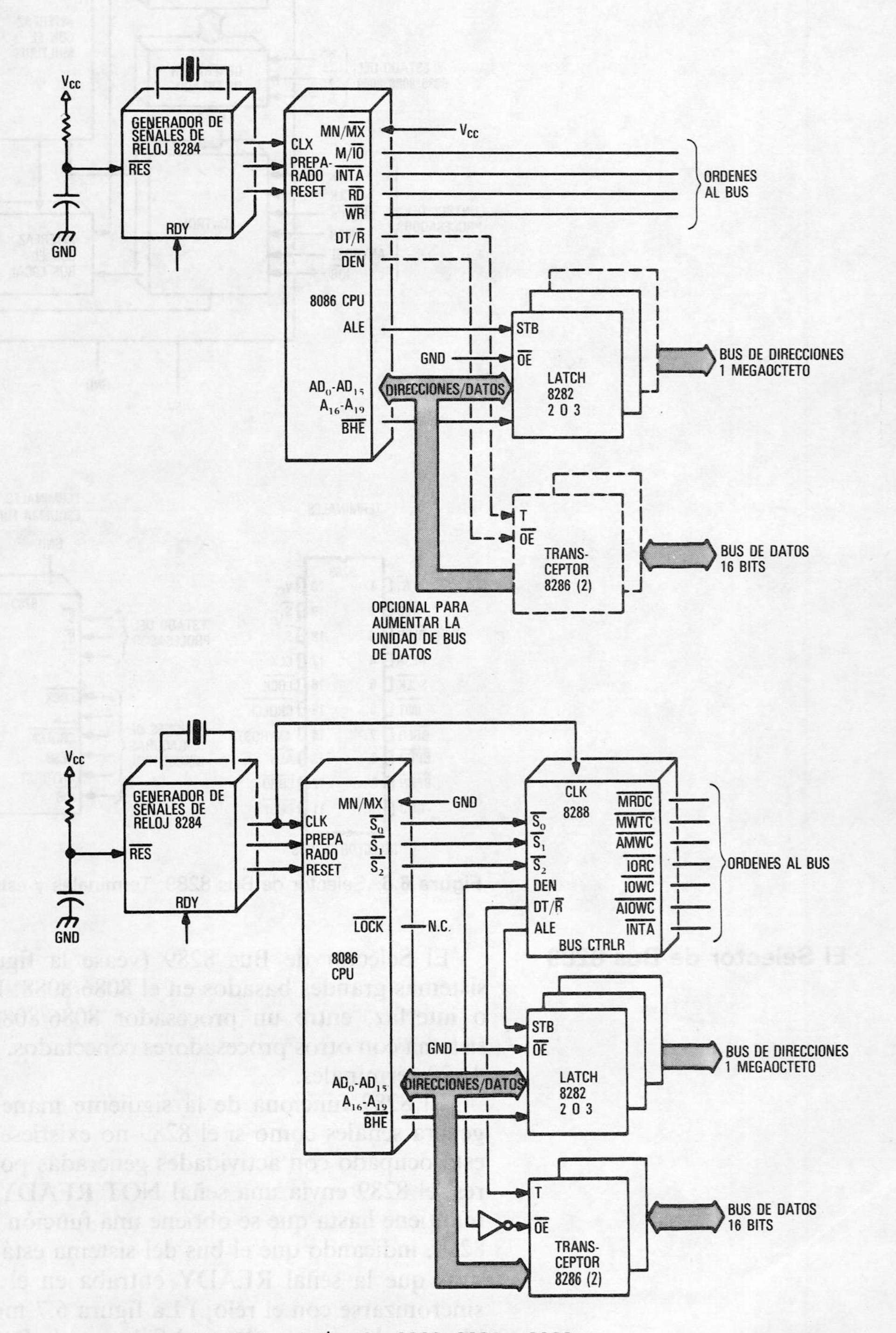
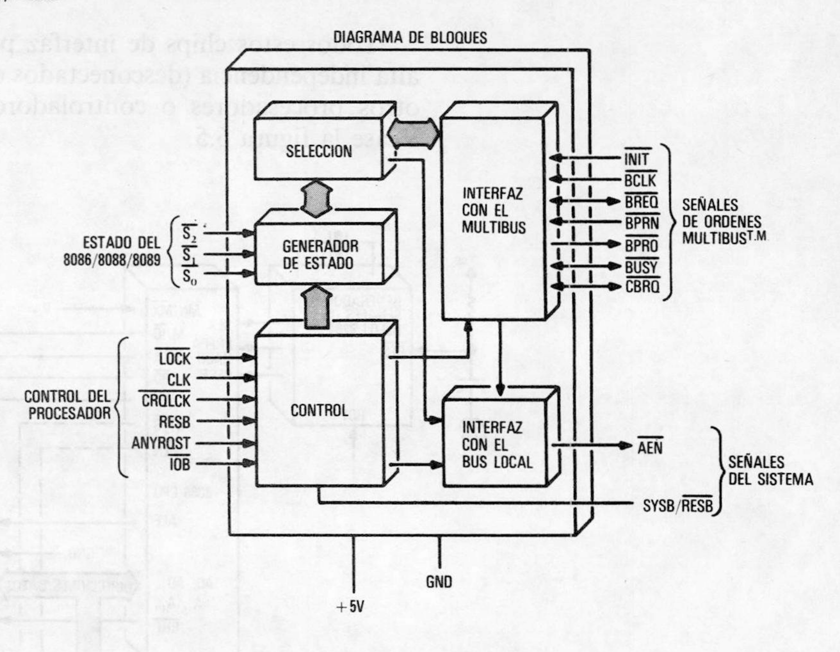


Figura 6.5: Chips de interfaz 8288, 8286 y 8282.



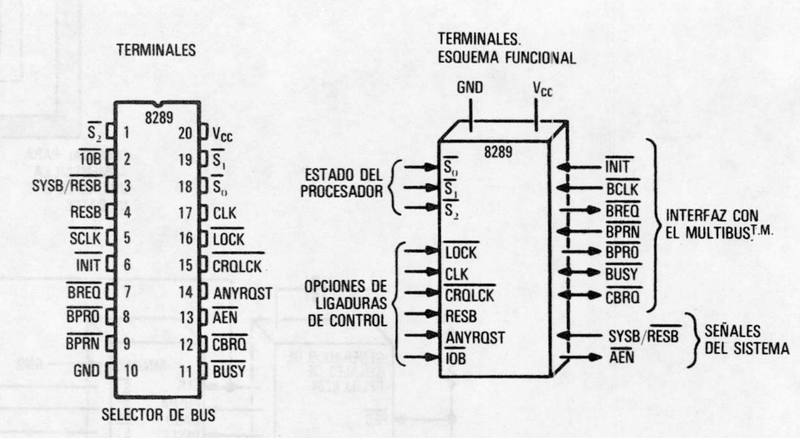


Figura 6.6: Selector de Bus 8289. Terminales y estructura interna.

El Selector de Bus 8289

El Selector de Bus 8289 (véase la figura 6.6) se utiliza en sistemas grandes basados en el 8086/8088. Realiza una conexión, o interfaz, entre un procesador 8086/8088/8089 y un bus del sistema con otros procesadores conectados. El 8289 viene en chips de 20 terminales.

El 8289 funciona de la siguiente manera: El 8086/8088/8089 genera señales como si el 8289 no existiese. Si el bus del sistema está ocupado con actividades generadas por los otros procesadores, el 8289 envía una señal NOT READY al procesador que se mantiene hasta que se obtiene una función READY de vuelta del 8289, indicando que el bus del sistema está disponible. (Recordemos que la señal READY entraba en el Generador 8284 para sincronizarse con el reloj.) La figura 6.7 muestra un diagrama de bloques de cómo utilizar el Selector de Bus 8289 en un circuito.

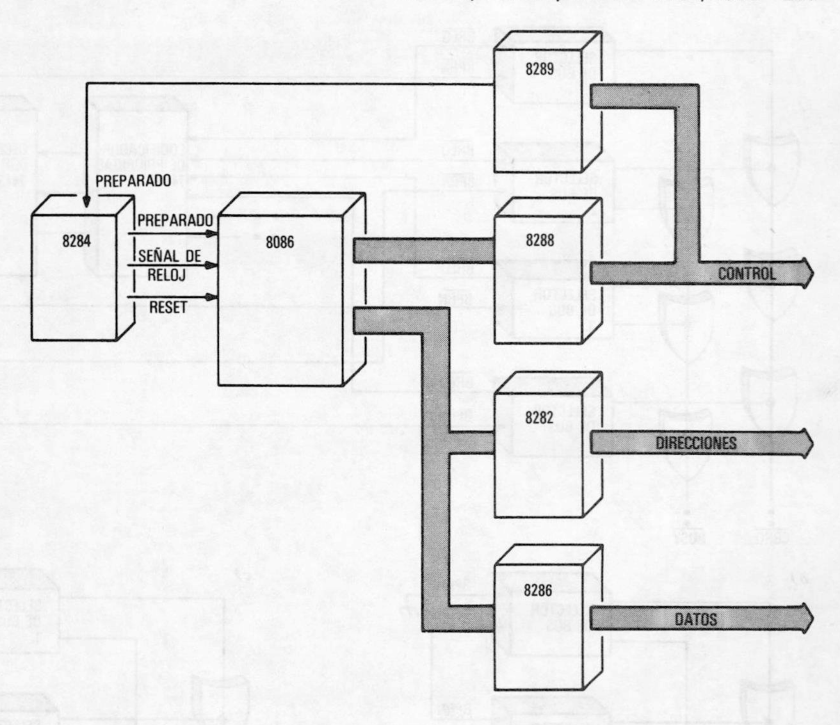


Figura 6.7: Diagrama de bloques mostrando cómo utilizar el selector de bus.

El Selector de Bus 8289 tiene varios esquemas de funcionamiento programables por el usuario, para decidir en cada momento qué procesador tiene acceso al bus principal. Los tres métodos básicos son: *prioridad paralela, prioridad serie* y *prioridad rotatoria*. Cada uno de estos esquemas involucran a más de un selector de bus, en concreto uno para cada grupo de procesadores.

Con el esquema de prioridad paralela, cada selector envía una señal de petición de bus a un circuito central de prioridades. Este circuito está inicializado de manera que asigna una prioridad distinta a cada grupo. Cuando recibe una señal de petición, determina qué petición de bus tiene mayor prioridad y permite al selector de bus correspondiente que tome control del bus.

Con el esquema de prioridad serie, los selectores de bus están conectados unos tras otros, como las luces de un árbol de Navidad, a través de líneas de señal. El primer selector de bus de la serie es el que tiene mayor prioridad, y el último es el que tiene menos. El terminal de salida de prioridad de bus de cada selector (línea BPRO), está conectado al de entrada de prioridad de bus del siguiente en secuencia (línea BPRN). El sistema serie no necesita circuitería adicional.

Con el esquema de prioridad rotatoria, las líneas de señal de cada selector de bus van a parar a una circuitería central, al igual que en el esquema de prioridad paralela. Pero esta vez, sin embargo, hay un esquema *dinámico:* más complejo que se encarga de determinar quién debe tener el control del bus.

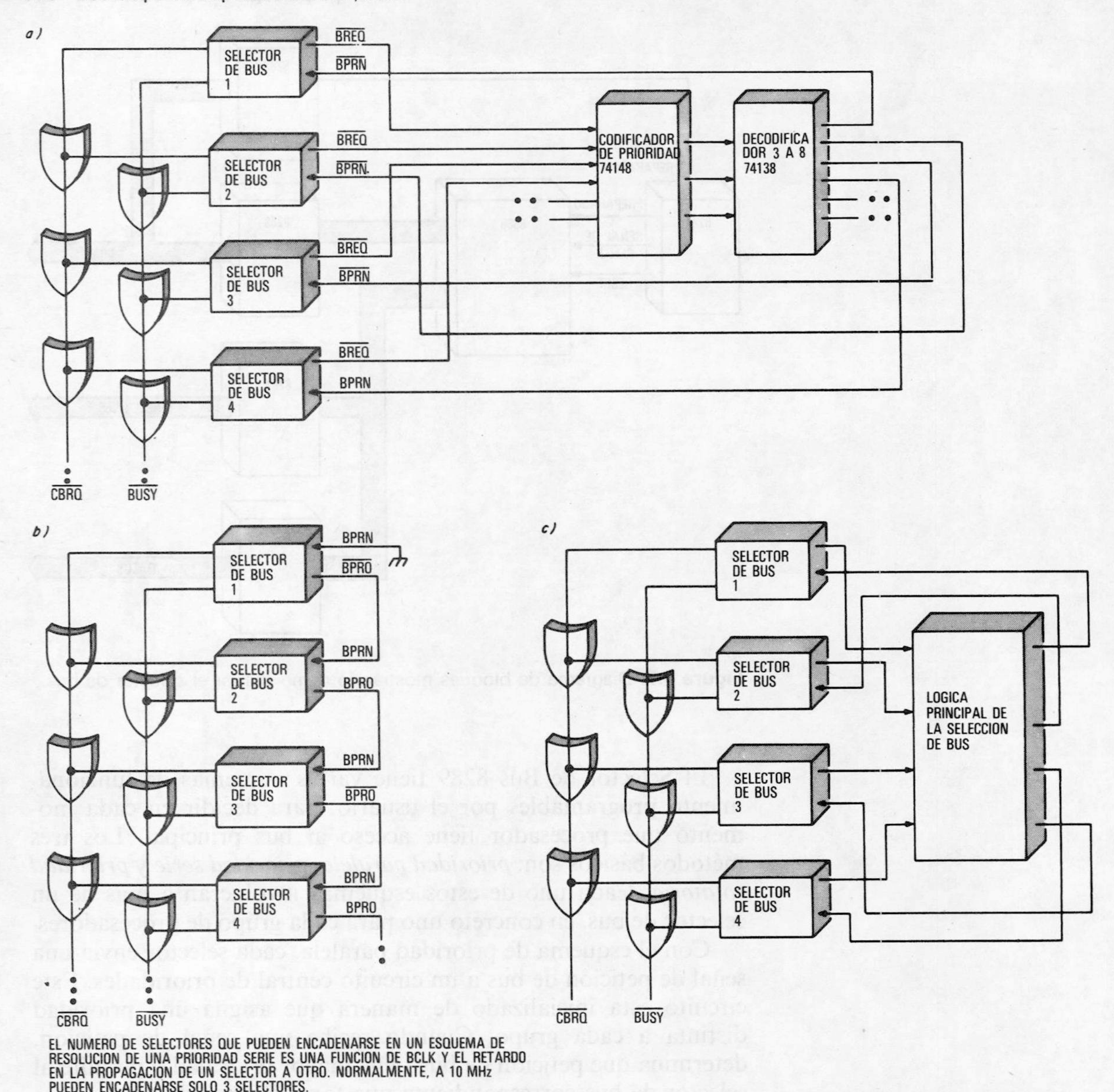


Figura 6.8a, b, c: Tres esquemas de selección de bus.

CONTROLADORES DE DISPOSITIVOS Y DEL SISTEMA

Hay dos tipos de controladores en un sistema de computadora; los controladores del sistema y los controladores de dispositivos.

Los controladores del sistema se encargan de controlar automáticamente funciones especiales del sistema como la transferencia rápida de bloques de datos, o la coordinación de varias interrupciones. En esta sección estudiaremos el Controlador Programable de DMA 8237, que realiza la función de transferencia rápida, y el controlador programable de interrupciones 8259, que controla, como su nombre indica, las interrupciones.

Los controladores de dispositivos hacen de interfaz inteligente con los dispositivos externos como unidades de discos, teclados e impresoras. Se pueden dividir en dos grupos: los controladores de dispositivos de propósito general y los de propósito especial. Aquí estudiaremos en concreto los controladores de dispositivos de propósito general 8251 (Controlador Programable Serie de Interfaz) y 8255 (Controlador Programable Paralelo de Interfaz). Ambos pueden conectarse a la variedad de dispositivos de la computadora. Como ejemplo de controladores de propósito especial veremos también el Controlador Programable de CRT 8275 y el Controlador de Discos Flexibles de Densidad Simple/-Doble 8272.

El Controlador Programable de DMA 8237

DMA son las siglas de «Direct Access Memory», esto es, Acceso Directo a Memoria. El Controlador de DMA es un dispositivo capaz de controlar las transferencias directas de información de una parte a otra del sistema. Dichas transferencias resultan muy interesantes por cuanto muchas veces es necesario mover bloques de datos muy rápidamente, algunas veces a velocidades mayores que las que se podrían conseguir moviendo los datos octeto a octeto, a través de la CPU. Por ejemplo, para visualizar figuras sobre una pantalla se necesita un barrido completo de pantalla (una *imagen)* 30 veces por segundo, y medio barrido (un *campo)* cada 1/60 segundos. (Los dos mediobarridos —campos— están entrelazados —entretejidos—.) Supongamos que queremos visualizar una figura en blanco y negro de 256 puntos horizontales por 240 verticales que cada uno de ellos puede estar a 0 o a 1. Una figura de *resolución media* como ésta se visualiza realizando un barrido para cada campo y dos por imagen. De esta manera cada punto aparece como un punto doble en pantalla, y la imagen completa se dibuja 60 veces cada segundo. Ahora bien, los 256×240 puntos requieren 61.440 bits de información; o lo que es lo mismo, 7680 octetos. Para visualizarlos 60 veces por segundo sería necesario poder procesar 1 octeto aproximadamente cada 2 microsegundos (1/60=16,67 microsegundos por barrido, que dividido por 7.680 da los aproximadamente 2 microsegundos por octeto). Cada vez que se explora el octeto tiene que obtenerse de la memoria. Sólo esto, con un reloj de 5 MHz, requiere ya unos 10 ciclos de reloj por octeto. Como la CPU 8086/8088 necesita alrededor de 17 ciclos para mover un octeto de información, le resulta totalmente imposible. Incluso si la CPU pudiese mover 16 bits a la vez (2 octetos), la mitad del tiempo estaría enviando bits a a la pantalla, lo que representaría una completa pérdida de tiempo para la CPU.

Para que la computadora funcione eficazmente, se necesitaría una circuitería especial encargada de leer estos octetos. Una solución es almacenar estos octetos en una memoria especial de visualización que llevara incorporada circuitería de barrido y un *esquema de selección* entre los accesos a memoria de la circuitería de barrido y los accesos a memoria de la CPU. Es lo que frecuentemente recibe el nombre de *almacenamiento temporal* o

memoria intermedia (buffer) marco. La memoria de un sistema como ésta se dice que es de *acceso-dual* (dual-ported), debido a que hay dos formas distintas de acceder a ella.

En contraste con este sistema, uno de los primeros de bajo-coste en aparecer (el Dazzler de Cromemco) guardaba dichos octetos en una memoria sencilla (de acceso único) y la circuitería de barrido consistía fundamentalmente en un dispositivo controlador de DMA que tomaba posesión del bus y generaba su propia dirección y la información de control sobre el bus del sistema. El Dazzler tenía en realidad la tercera parte de resolución del ejemplo y funcionaba a 2 MHz, pero presentaba problemas de temporización similares a los del ejemplo.

Debido a las restricciones impuestas por la temporización, un DMA como acabamos de describir no parece la mejor solución para el barrido directo de una memoria regular de un visualizador, pero es un método bueno para conseguir transferencias de información rápidas. El DMA se usa, por ejemplo, cuando se quiere mover rápidamente una imagen, o parte de una imagen, de la memoria intermedia marco a la memoria; o cuando se necesita transferir rápidamente los contenidos de las memorias intermedias de un disco flexible a otras posiciones.

En general, un controlador DMA se usa de la siguiente manera: Se le dice al Controlador de DMA que se va a hacer una transferencia o bien por medio de la CPU o por alguna circunstancia especial; entonces el Controlador de DMA pide el control del bus. Si la CPU, otros procesadores, u otros controladores estaban utilizando el bus en aquel momento, lo liberan poniendo sus líneas de conexión en el estado de alta impedancia (desconectadas): entonces pasan el control del bus al controlador de DMA, y finalmente, el controlador toma posesión del bus, genera su propia dirección y las señales de control necesarias para provocar la transferencia de información. Véase la figura siguiente:

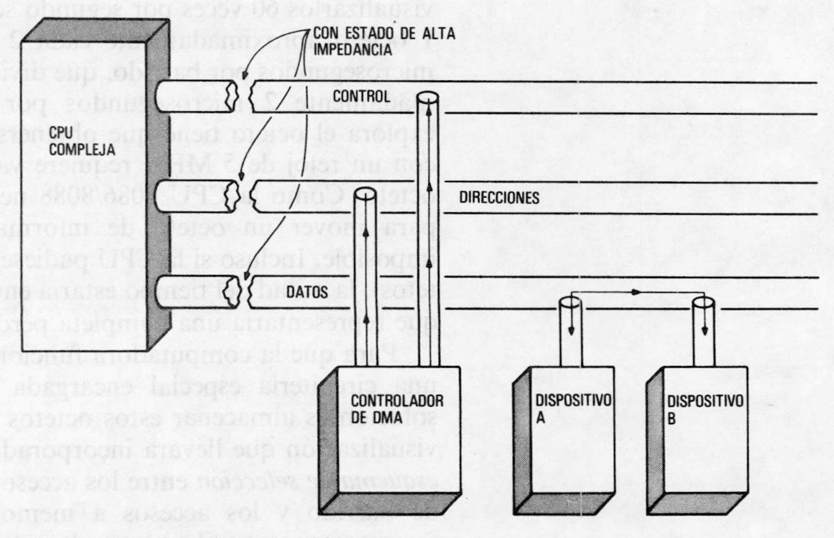


Figura 6.9: Transferencia DMA.

El Controlador de DMA Intel 8237 que puede verse en la figura 6.10 se utiliza en las transferencias por DMA. Viene en un chip de 40 terminales, y puede dar servicio a un total de cuatro dispositivos distintos a la vez. Por ejemplo, un Controlador de DMA 8237 puede gestionar a la vez las transferencias de dos visualizadores CRT, un controlador de discos flexibles y una unidad de cinta magnética.

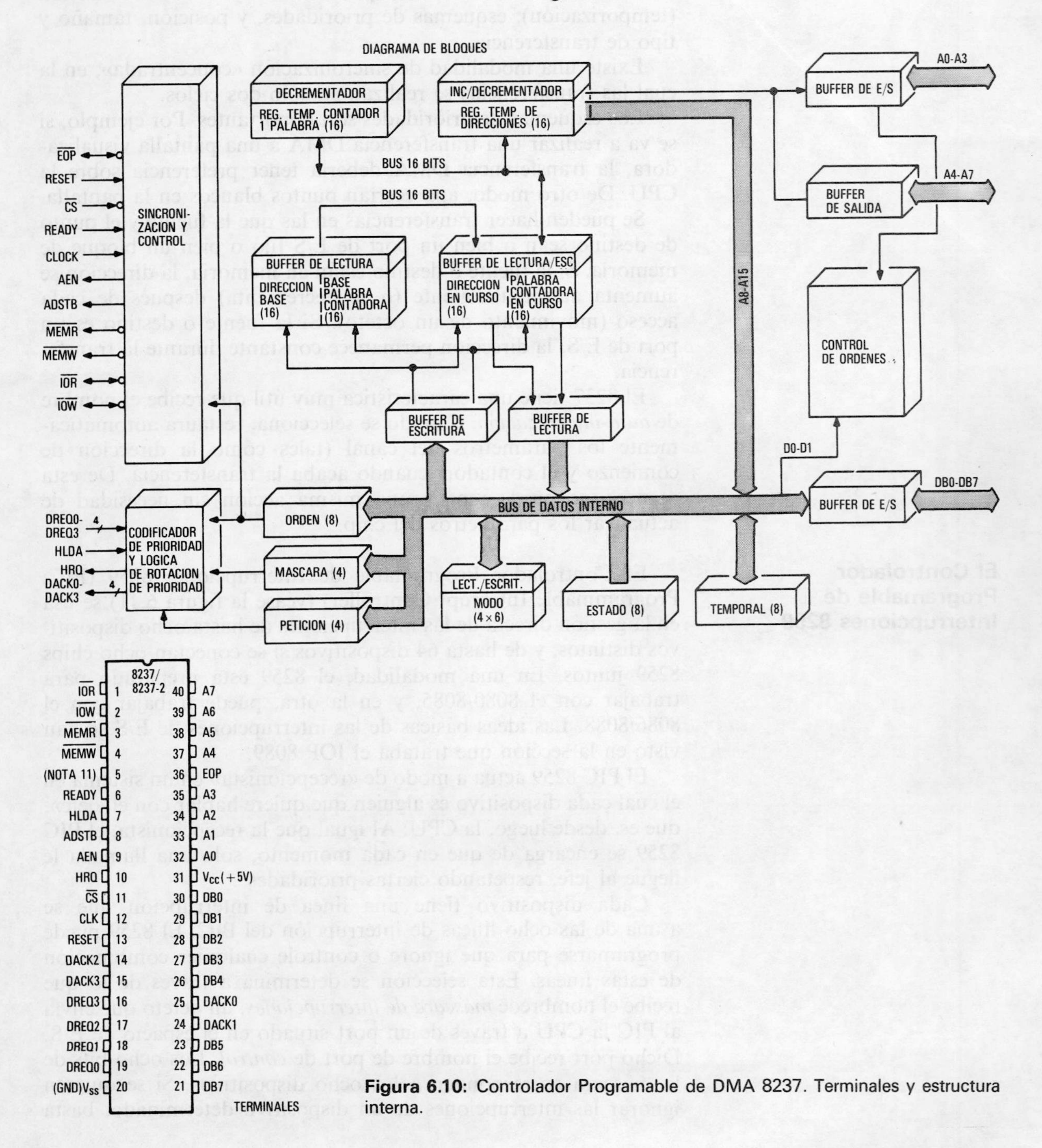


Figura 6.10: Controlador Programable de DMA 8237. Terminales y estructura interna.

A cada dispositivo se le asigna un *canal* del 8237. Interconectando varios chips DMA se puede soportar los canales que se deseen. El 8237 posee registros encargados de guardar la información sobre las direcciones fuente y destino, contadores y máscaras, órdenes y estado. El 8237 se programa llevando octetos de E/S a registros especiales de órdenes y de modo de funcionamiento. Tiene disponibles muchas opciones, incluyendo sincronización (temporización); esquemas de prioridades, y posición, tamaño y tipo de transferencia.

Existe una modalidad de sincronización «concentrada», en la cual las transferencias se realizan en sólo dos ciclos.

Los esquemas de prioridades son importantes. Por ejemplo, si se va a realizar una transferencia DMA a una pantalla visualizadora, la transferencia DMA debería tener preferencia sobre la CPU. De otro modo, aparecerían puntos blancos en la pantalla.

Se pueden hacer transferencias en las que la fuente y el punto de destino sean o bien un port de E/S fijo o bien un bloque de memoria. Si la fuente o destino están en memoria, la dirección se aumenta automáticamente (o se decrementa) después de cada acceso (movimiento de un octeto). Si la fuente o destino es un port de E/S, la dirección permanece constante durante la transferencia.

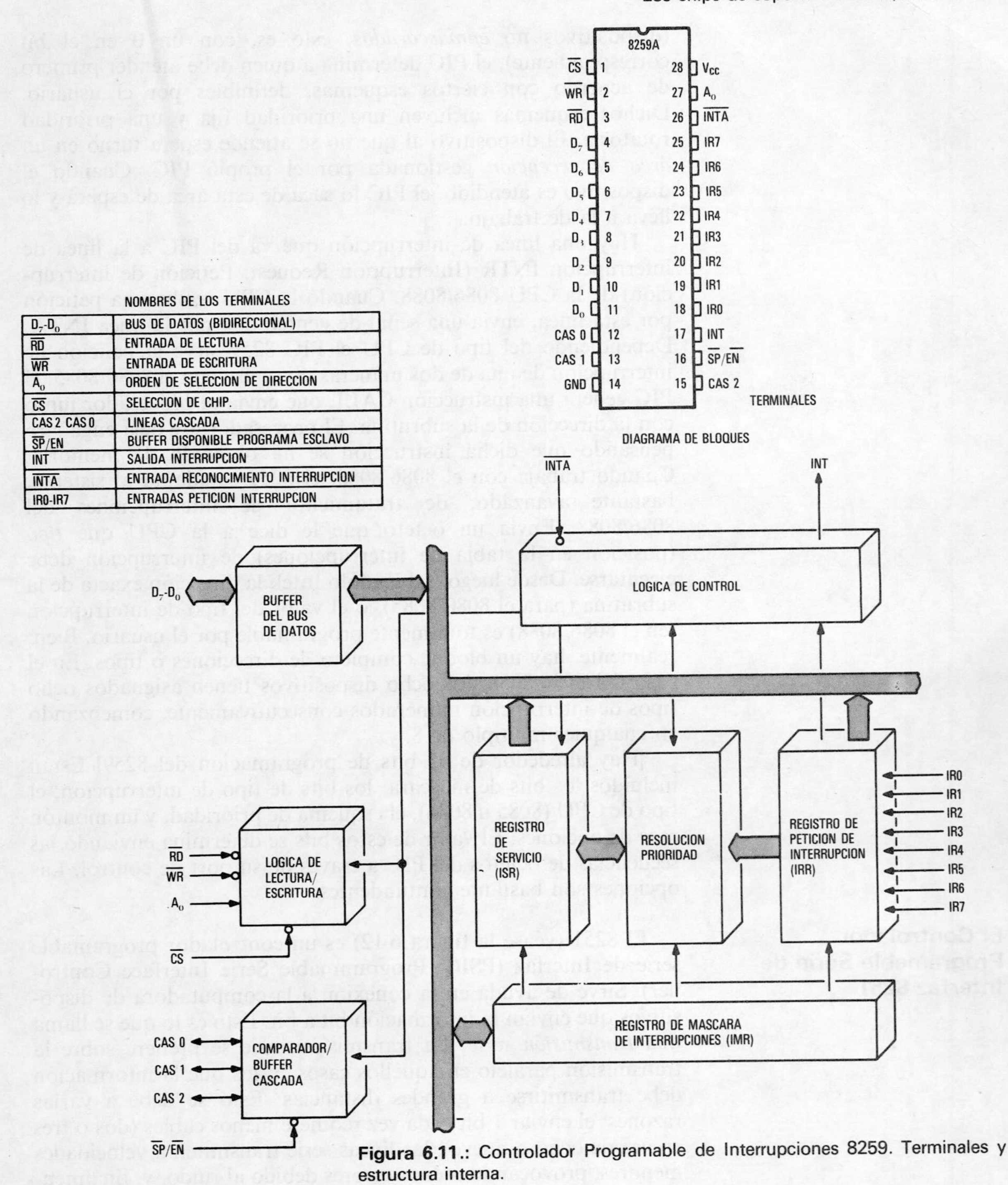
El 8237 tiene una característica muy útil que recibe el nombre de *auto-inicialización*. Cuando se selecciona, restaura automáticamente los parámetros del canal (tales como la dirección de comienzo y el contador) cuando acaba la transferencia. De esta manera, se puede repetir una misma acción sin necesidad de actualizar los parámetros del chip.

El Controlador Programable de Interrupciones 8259

El Controlador Programable de Interrupciones 8259 (PIC: Programmable Interrupt Controller) (véase la figura 6.11) se usa en la gestión directa de las interrupciones de hasta ocho dispositivos distintos, y de hasta 64 dispositivos si se conectan ocho chips 8259 juntos. En una modalidad, el 8259 está preparado para trabajar con el 8080/8085, y en la otra, puede trabajar con el 8086/8088. Las ideas básicas de las interrupciones de E/S se han visto en la sección que trataba el IOP 8089.

El PIC 8259 actúa a modo de «recepcionista» en un sistema en el cual cada dispositivo es alguien que quiere hablar con el «jefe», que es, desde luego, la CPU. Al igual que la recepcionista, el PIC 8259 se encarga de que en cada momento, sólo una llamada le llegue al jefe, respetando ciertas prioridades.

Cada dispositivo tiene una línea de interrupción que se a una de las ocho líneas de interrupción del PIC. El 8259 puede programarse para que ignore o controle cualquier combinación de estas líneas. Esta selección se determina a través de lo que recibe el nombrede *máscara de interrupciones,* un octeto que envía al PIC la CPU a través de un port situado en el espacio de E/S. Dicho port recibe el nombre de port de *control.* Los ocho bits de la máscara corresponden a los ocho dispositivos. Si se quieren ignorar las interrupciones de un dispositivo determinado, basta



NOMBRES DE LOS TERMINALES

D_7-D_0	BUS DE DATOS (BIDIRECCIONAL)
$\overline{RD}$	ENTRADA DE LECTURA
$\overline{WR}$	ENTRADA DE ESCRITURA
A_0	ORDEN DE SELECCION DE DIRECCION
$\overline{CS}$	SELECCION DE CHIP
CAS 2 CAS 0	LINEAS CASCADA
$\overline{SP}$/$\overline{EN}$	BUFFER DISPONIBLE PROGRAMA ESCLAVO
INT	SALIDA INTERRUPCION
$\overline{INTA}$	ENTRADA RECONOCIMIENTO INTERRUPCION
IR0-IR7	ENTRADAS PETICION INTERRUPCION

Figura 6.11.: Controlador Programable de Interrupciones 8259. Terminales y estructura interna.

con poner a 1 el bit de la máscara de interrupciones asociado a dicho dispositivo. Así, si la máscara es 11111111B, el PIC ignorará las interrupciones *provenientes de todos los dispositivos*, y si la máscara es 00000000B, responderá a todos los dispositivos.

Si al PIC le llegan señales de dos o más dispositivos a la vez

(dispositivos no *enmascarados,* esto es, con un 0 en el bit correspondiente), el PIC determina a quién debe atender primero de acuerdo con ciertos esquemas, definibles por el usuario. Dichos esquemas incluyen una prioridad fija y una prioridad rotatoria. El dispositivo al que no se atiende espera turno en un *área de recepción* gestionada por el propio PIC. Cuando el dispositivo es atendido, el PIC lo saca de esta área de espera y lo lleva a la de trabajo.

Hay una línea de interrupción que va del PIC a la línea de interrupción INTR (Interruption Request: Petición de interrupción) de la CPU 8086/8088. Cuando la CPU recibe una petición por esta línea, envía una señal de «enterado» por la línea INTA. Dependiendo del tipo de CPU el PIC 8259 sirve la petición de interrupción de una de dos maneras. Si se usa con el 8080/8085, el PIC genera una instrucción CALL que envía al procesador junto con la dirección de la subrutina. El procesador «se deja engañar» pensando que dicha instrucción se ha obtenido de memoria. Cuando trabaja con el 8086/8088, el 8259 se adapta al sistema, bastante avanzado, de tratamiento de interrupciones del 8086/8088. Envía un octeto que le dice a la CPU qué *tipo* (posición en la tabla de interrupciones) de interrupción debe ejecutarse. Desde luego, sabiéndolo Intel, la dirección exacta de la subrutina (para el 8080/8085), o el valor del tipo de interrupción (en el 8086/8088) es totalmente programable por el usuario. Bien, realmente, hay un bloque completo de direcciones o tipos. En el caso del 8086/8088, los ocho dispositivos tienen asignados ocho tipos de interrupción numerados consecutivamente, comenzando en cualquier múltiplo de 8.

¡Hay alrededor de 45 bits de programación del 8259! Están incluidos los bits de máscara, los bits de tipo de interrupción, el tipo de CPU (8085 u 8086), el esquema de prioridad, y un montón más de opciones. El valor de estos bits se determina enviando las secuencias de octetos del PIC a través de su port de control. Las opciones son bastante contundentes.

El Controlador Programable Serie de Interfaz 8251

El 8251 (véase la figura 6.12) es un controlador programable serie de Interfaz (PSIC: Programmable Serie Interface Controller). Sirve de ayuda en la conexión a la computadora de dispositivos que envían la información bit a bit. Esto es lo que se llama una *transmisión serie*. La transmisión serie se prefiere sobre la transmisión paralelo en aquellos casos en los que la información debe transmitirse a grandes distancias. Esto se debe a varias razones: el enviar 1 bit cada vez requiere menos cables (dos o tres en vez de ocho o nueve); las líneas serie transmiten a velocidades menores, provocando menos errores debido al ruido, y, finalmente, las transmisiones serie son fácilmente codificables y decodificables para su envío por líneas telefónicas. Para ello se usa un modem.

Puesto que la computadora trabaja internamente con los datos en paralelo, se necesita un controlador serie de interfaz para pasar de paralelo a serie. Un dispositivo de este tipo recibe el

NOMBRE DE LOS TERMINALES

D_7-D_0	BUS DE DATOS (8 BITS)	$\overline{DSR}$	DATOS PREPARADOS
C/$\overline{D}$	SE VAN A LEER O ESCRIBIR CONTROL O DATOS	$\overline{DTA}$	TERMINAL DE DATOS PREPARADO
$\overline{RD}$	ORDEN DE LECTURA DE DATOS	SYNDET-BD	DETECTADO SYNC/ DETECTADA RUPTURA
$\overline{WR}$	ESCRIBIR DATOS U ORDEN DE CONTROL		
$\overline{CS}$	HABILITACION CHIP	$\overline{RTS}$	PETICION DE ENVIO DE DATOS
CLK	PULSO DE RELOJ (TTL)	$\overline{CTS}$	BORRAR PARA ENVIAR DATOS
RESET	BORRADO	TxE	TRANSMISOR VACIO
$\overline{TxC}$	TRANSMISOR SEÑALES DE RELOJ	V_{CC}	ALIMENTACION +5 VOLTS.
TxD	TRANSMISOR DATOS	GND	TIERRA
$\overline{RxC}$	RECEPTOR SEÑALES DE RELOJ		
RxD	RECEPTOR DATOS		
RxRDY	RECEPTOR PREPARADO (CONTIENE CARACTER PARA 8080)		
TxRDY	TRANSMISOR PREPARADO (PREPARADO PARA CARACTER DEL 8080)		

TERMINALES

8251A

D_2	1	28	D_1
D_3	2	27	D_0
RxD	3	26	V_{CC}
GND	4	25	$\overline{RxC}$
D_4	5	24	$\overline{DTR}$
D_5	6	23	$\overline{RTS}$
D_6	7	22	$\overline{DSR}$
D_7	8	21	RESET
$\overline{TxC}$	9	20	CLK
$\overline{WR}$	10	19	TxD
$\overline{CS}$	11	18	TxEMPTY
C/$\overline{D}$	12	17	$\overline{CTS}$
$\overline{RD}$	13	16	SYNDET/BD
RxRDY	14	15	TxRDY

DIAGRAMA DE BLOQUES

D_7-D_0
BUFFER DEL BUS DE DATOS
BUFFER DE TRANSMISION (P-S)
TxD
RESET
CLK
C/$\overline{D}$
$\overline{RD}$
$\overline{WR}$
LOGICA DE CONTROL DE LECTURA/ ESCRITURA
$\overline{CS}$
CONTROL DE TRANSMISION
TxRDY
TxE
$\overline{TxC}$
$\overline{DSR}$
$\overline{DTR}$
$\overline{CTS}$
$\overline{RTS}$
CONTROL DEL MODEM
BUFFER DE RECEPCION (S-P)
RxD
BUS DE DATOS INTERNO
CONTROL DE RECEPCION
RxRDY
$\overline{RxC}$
SYNDET

Figura 6.12: Controlador Programable Serie de Interfaz 8251. Terminales y estructura interna.

nombre de receptor/transmisor asíncrono universal (UART: Universal Asynchronous Receiver/Transmitter). El 8251 es en realidad un receptor/trasmisor síncrono/asíncrono universal (USART). La diferencia estriba en que el 8251 puede operar en modo síncrono o asíncrono. En modo asíncrono, cada octeto se trata por separado. Cuando hay un octeto preparado para su transmisión, se envían 1 ó 2 bits indicando que los bits siguientes pertenecen a un octeto de información. Estos bits reciben el nombre de *bits de inicio*. Cuando se han enviado todos los bits del octeto normalmente, se acaba la transmisión con unos *bits de final*. Se suelen necesitar 10 bits para enviar un carácter de esta

manera. En el modo síncrono, todo un bloque de octetos se envía como una unidad. Hay un formato de inicio especial, pero no se envían bits extra con cada octeto. El modo síncrono es más rápido, pero necesita más software. El trabajo extra que produce lo hace eficiente sólo cuando hay que enviar grandes lotes de información. En ambos modos, los bits individuales se envían a una cierta frecuencia o velocidad, llamada velocidad en baudios. Son bastante estándar las velocidades de transmisión de 110, 300, 1.200, 2.400, 4.800, 9.600 y 19.200 bits por segundo.

El PSIC 8251 viene encapsulado en chips de 28 terminales. Debe inicializarse con una secuencia de octetos que se suele enviar vía el bus del sistema al espacio de E/S. El usuario puede seleccionar el tipo de transmisión síncrono o asíncrono. En las transmisiones asíncronas, el usuario puede definir también el fomato de los caracteres a enviar, incluyendo el número de bits de inicio, de bits de final y el tipo de paridad a controlar (o la no paridad). Se puede incluso controlar la velocidad de transmisión. Cuando se usa junto al Temporizador de Intervalo Programable 8253 (Programmable Interval Timer), el PSIC 8251 puede ajustar su velocidad a cualquier velocidad que utilice el emisor (el cómo se puede conseguir esto no lo trataremos aquí). La figura 6.13 muestra cómo programar el PSIC 8251.

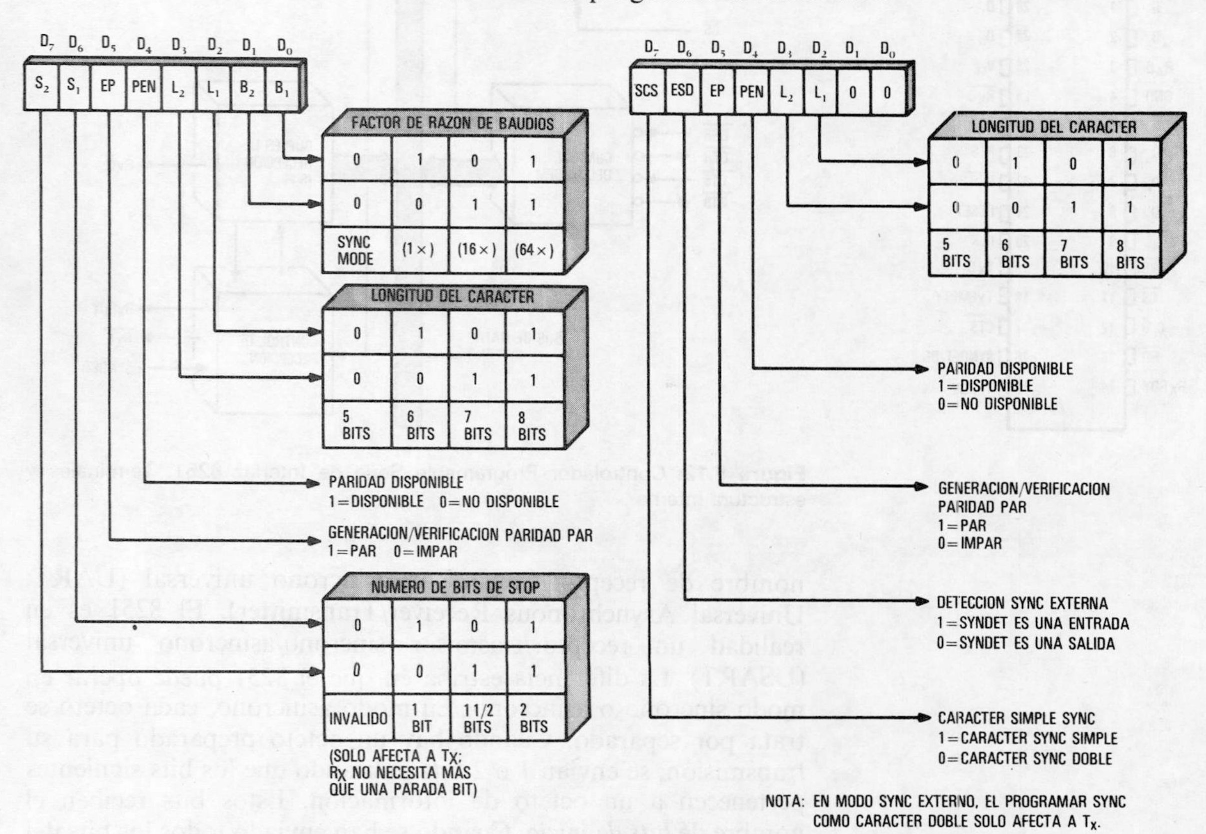
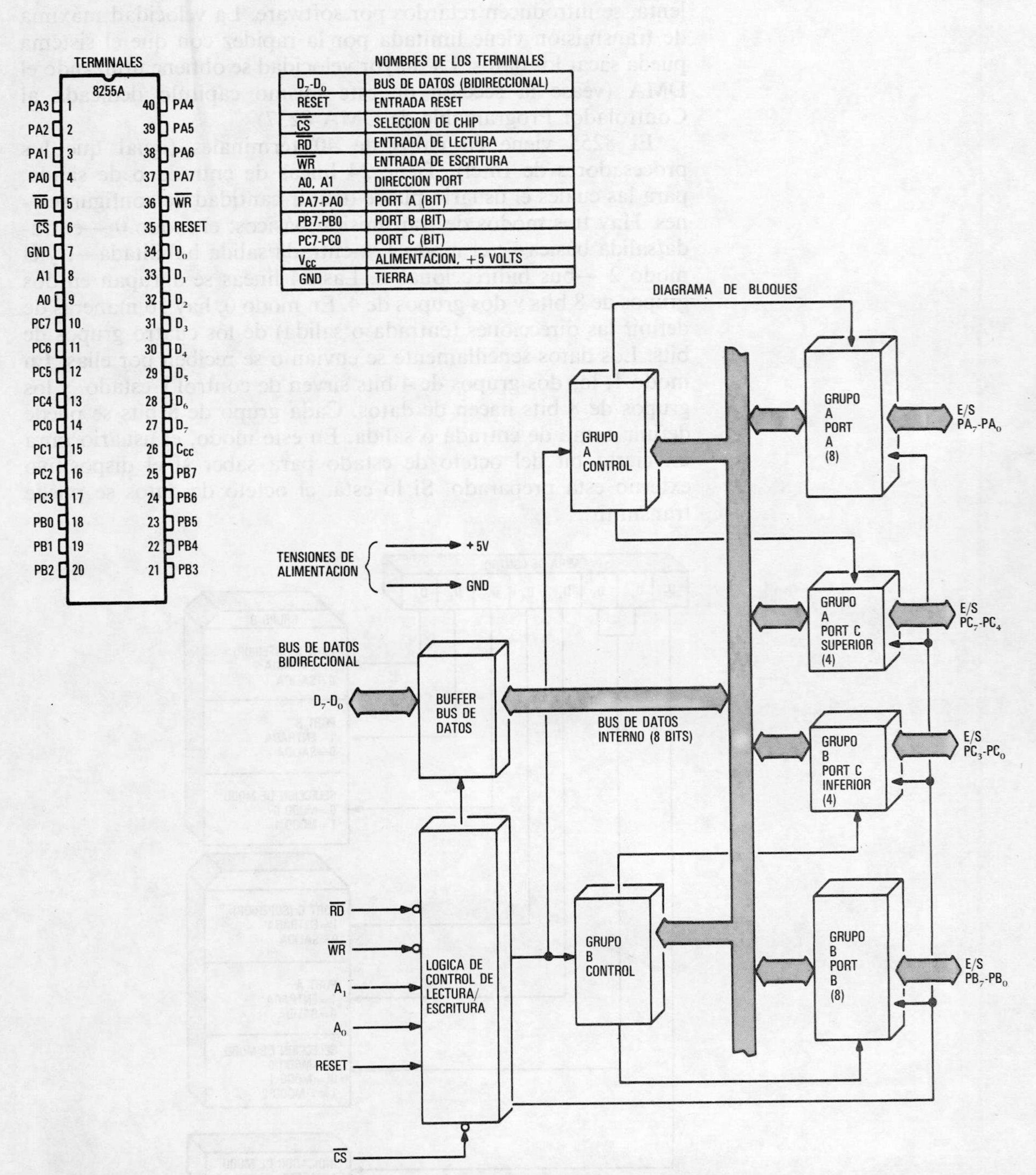


Figura 6.13: Programación del PSIC 8251.

El Controlador Programable Paralelo de Interfaz 8255

El Controlador Programable Paralelo de Interfaz 8255 (PPIC: Programable Paradell Interface Controller) (véase la figura 6.14) sirve de ayuda en la conexión a la computadora de dispositivos que envían octetos completos cada vez (o incluso palabras de 12, 16 ó 24 bits). La transmisión paralelo es útil en todas aquellas



NOMBRES DE LOS TERMINALES	
D_7-D_0	BUS DE DATOS (BIDIRECCIONAL)
RESET	ENTRADA RESET
$\overline{CS}$	SELECCION DE CHIP
$\overline{RD}$	ENTRADA DE LECTURA
$\overline{WR}$	ENTRADA DE ESCRITURA
A0, A1	DIRECCION PORT
PA7-PA0	PORT A (BIT)
PB7-PB0	PORT B (BIT)
PC7-PC0	PORT C (BIT)
V_{CC}	ALIMENTACION, +5 VOLTS
GND	TIERRA

Figura 6.14: Controlador Programable Paralelo de Interfaz 8255. Terminales y estructura interna.

aplicaciones que requieran unas transmisiones a gran velocidad, y utilicen dispositivos no demasiado alejados del computador central. No hay ninguna sincronización especial en las transmisiones paralelo. Los octetos se envían tan rápidamente, o tan lentamente, como permite el software. Si la transmisión debe ser lenta, se introducen retardos por software. La velocidad máxima de transmisión viene limitada por la rapidez con que el sistema pueda sacar los datos. La mayor velocidad se obtiene utilizando el DMA (véase la sección de este mismo capítulo dedicada al Controlador Programable de DMA 8237).

El 8255 viene en chips de 40 terminales (igual que los procesadores de Intel). Tiene 24 líneas de entrada o de salida, para las cuales el usuario puede definir cantidad de configuraciones. Hay tres modos de transmisión básicos: el modo 0 —entrada/salida básica—; el modo 1 —entrada/salida habilitada—, y el modo 2 —bus bidireccional—. Las 24 líneas se agrupan en dos grupos de 8 bits y dos grupos de 4. En modo 0, hay 16 maneras de definir las direcciones (entrada o salida) de los cuatro grupos de bits. Los datos sencillamente se envían o se reciben por ellas. En modo 1, los dos grupos de 4 bits sirven de control y estado, y los grupos de 8 bits hacen de datos. Cada grupo de 8 bits se puede definir como de entrada o salida. En este modo, el usuario mira un cierto bit del octeto de estado para saber si el dispositivo externo está preparado. Si lo está, el octeto de datos se puede transmitir.

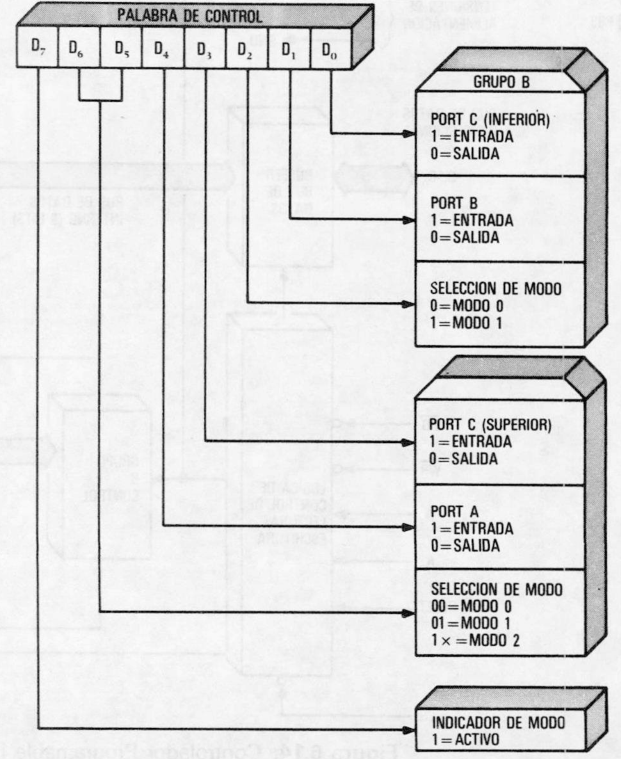


Figura 6.15: Programación del PPIC 8255.

Las órdenes se transmiten al 8255 vía un port de E/S especial. Estas órdenes afectan a características como el agrupamiento, la dirección de los ports, y las asignaciones de control y estado. La figura 6.15 muestra cómo funciona todo esto:

La figura 6.16 muestra el ejemplo de un programa que utiliza el PPIC 8255 y el PSIC 8251. El programa principal es un bucle que convierte al computador en terminal de otro computador, que podría estar a cierta distancia. La línea de comunicación con este computador utiliza la transmisión serie, de forma que se usa el PSIC 8251 como interfaz. El equipo local consta de un teclado y una pantalla de video. Ambos dispositivos están conectados a la computadora local vía un PPIC 8255. El teclado y la pantalla trabajan a modo de consola para el sistema total.

```
                           ; BUCLE TERMINAL
                           ; **************
ØØØØ'   E8 ØØ1E'           TERM:    CALL    CINIT        ; INICIALIZAR EL PPI COMO PLOTTER
ØØØ3'   E8 ØØ24'                    CALL    XINIT        ; INICIALIZAR LA USART PARA E/S EXTERNA
ØØØ6'   E8 ØØ32'           TERMØ:   CALL    CINP         ; ESPERAR ENTRADA DE CONSOLA
ØØØ9'   74 Ø7                       JZ      TERM1        ;   SI SE ENCUENTRA, IR A TERM1
ØØØB'   E8 ØØ4D'                    CALL    XINP         ; ESPERAR ENTRADA EXTERNA
ØØØE'   74 Ø9                       JZ      TERM2        ;   CUANDO SE ENCUENTRA, IR A TERM2
ØØ1Ø'   EB F4                       JMPS    TERMØ        ; CONTINUA ESPERANDO ENTRADA
ØØ12'   8A EØ              TERM1:   MOVB    AH,AL
ØØ14'   E8 ØØ53'                    CALL    XOUT         ; TRANSMITIR ENTRADA DE CONSOLA
ØØ17'   EB ED                       JMPS    TERMØ
ØØ19'   8A EØ              TERM2:   MOVB    AH,AL
ØØ1B'   E8 ØØ2C'                    CALL    COUT         ; SI NO <ESC>, SALIDA POR CONSOLA
ØØ1E'   E9 FFE5'                    JMP     TERMØ        ; CONTINUA ESPERANDO ENTRADA
                           ;
                           ; INICIALIZACION DE CONSOLA (PPIC)
                           ; ********************************
ØØ21'   BØ B8              CINIT:   MOVI    AL,ØB8H      ; B8 HEX ES LA PALABRA DE CONTROL DE MODO
ØØ23'   E6 E7                       OUTB    ØE7H         ; ENVIAR A CONSOLA PPI
ØØ25'   BØ Ø3                       MOVBI   AL,Ø3        ; Ø3 ES EL BIT SET/RESET DE LA PALABRA DE CONTROL
ØØ27'   E6 E7                       OUTB    ØE7H         ; PONER A 1 EL BIT DE HABILITACION VIDEO
ØØ29'   C3                          RET
                           ;
                           ; INICIALIZACION DE E/S EXTERNA (USART)
                           ; *************************************
ØØ2A'   BØ AA              XINIT:   MOVBI   AL,ØAAH      ; ORDEN FICTICIA
ØØ2C'   E6 ED                       OUTB    ØEDH         ;   ENVIAR A LA USART
ØØ2E'   BØ 4Ø                       MOVBI   AL,4ØH       ; BORRAR ORDEN
ØØ3Ø'   E6 ED                       OUTB    ØEDH         ;   ENVIAR A LA USART
ØØ32'   BØ CE                       MOVBI   AL,ØCEH      ; INSTRUCCION DE MODO DE LA USART
ØØ34'   E6 ED                       OUTB    ØEDH         ; ENVIAR INSTRUCCION DE MODO A LA USART
ØØ36'   BØ 27                       MOVBI   AL,27H       ; ORDEN A TRANSMITIR/RECIBIR
ØØ38'   E6 ED                       OUTB    ØEDH         ; ENVIADA A LA USART
ØØ3A'   C3                          RET
                           ; RUTINA DE ENTRADA POR CONSOLA
                           ; *****************************
ØØ3B'   E4 E6              CINP:    INB     ØE6H         ; MIRAR EL PORT DE ESTADO DE LA CONSOLA
ØØ3D'   24 4Ø                       ANDBI   AL,4ØH
ØØ3F'   75 Ø8                       JNZ     CIEND        ; SI NO HAY ENTRADA ESPERANDO, VOLVER
ØØ41'   E4 E4                       INB     ØE4H         ; OBTENER LA ENTRADA DE CONSOLA
ØØ43'   F6 DØ                       NOTB    AL
ØØ45'   24 7F                       ANDBI   AL,7FH       ; SEPARAR BIT DE PARIDAD
ØØ47'   3A CØ                       CMPB    AL,AL        ; PONER A 1 EL BIT Z
ØØ49'   C3                 CIEND:   RET
                           ;

                           ; RUTINA DE SALIDA POR CONSOLA
                           ; ****************************
ØØ4A'   8A C4              COUT:    MOVB    AL,AH
ØØ4C'   E6 E5                       OUTB    ØE5H         ; SALIDA A VIDEO
ØØ4E'   BØ Ø2                       MOVBI   AL,Ø2
ØØ5Ø'   E6 E7                       OUTB    ØE7H         ; BORRAR DISPOSITIVO PPI
ØØ52'   FE CØ                       INCB    AL           ; PREPARAR DISPOSITIVO PPI PARA
```

```
0054'   E6 E7              OUTB    0E7H          ;   LA SIGUIENTE SALIDA
0056'   B1 00              MOVBI   CL,0          ; COMENZAR RETARDO
0058'   E2 FE     COUT1:   LOOP    COUT1
005A'   C3                 RET                   ; VOLVER TRAS EL RETARDO
                  ;
                  ; RUTINA DE ENTRADA EXTERNA
                  ; ************************
005B'   E4 ED     XINP:    INB     0EDH          ; VERIFICAR PALABRA DE ESTADO ENTRADA EXTERNA
005D'   F6 D0              NOTB    AL            ;   PARA EL INDICADOR DE ENTRADA ACTIVO
005F'   24 02              ANDBI   AL,02
0061'   75 06              JNZ     XIEND         ; SI NO HAY NINGUNO, VOLVER
0063'   E4 EC              INB     0ECH          ; LEER ENTRADA EXTERNA
0065'   24 7F              ANDBI   AL,7FH        ; LEER ENTRADA EXTERNA
0067'   3A C0              CMPB    AL,AL         ; AISLAR BIT DE PARIDAD
0069'   C3        XIEND:   RET
                  ;
                  ; RUTINA DE SALIDA EXTERNA
                  ; ***********************
006A'   E4 ED     XOUT:    INB     0EDH          ; VERIFICAR ESTADO DE SALIDA DE LA USART
006C'   24 01              ANDBI   AL,01         ;   SI NO ESTA PREPARADA PARA SALIDA
006E'   74 FA              JZ      XOUT          ;   CONTINUAR VERIFICANDO
0070'   8A C4              MOVB    AL,AH
0072'   E6 EC              OUTB    0ECH          ; TRANSMITIR CARACTER
0074'   C3                 RET
                  ;
                           END
```

Figura 6.16: Programa para terminal utilizando el PPIC 8255 y el PSIC 8251.

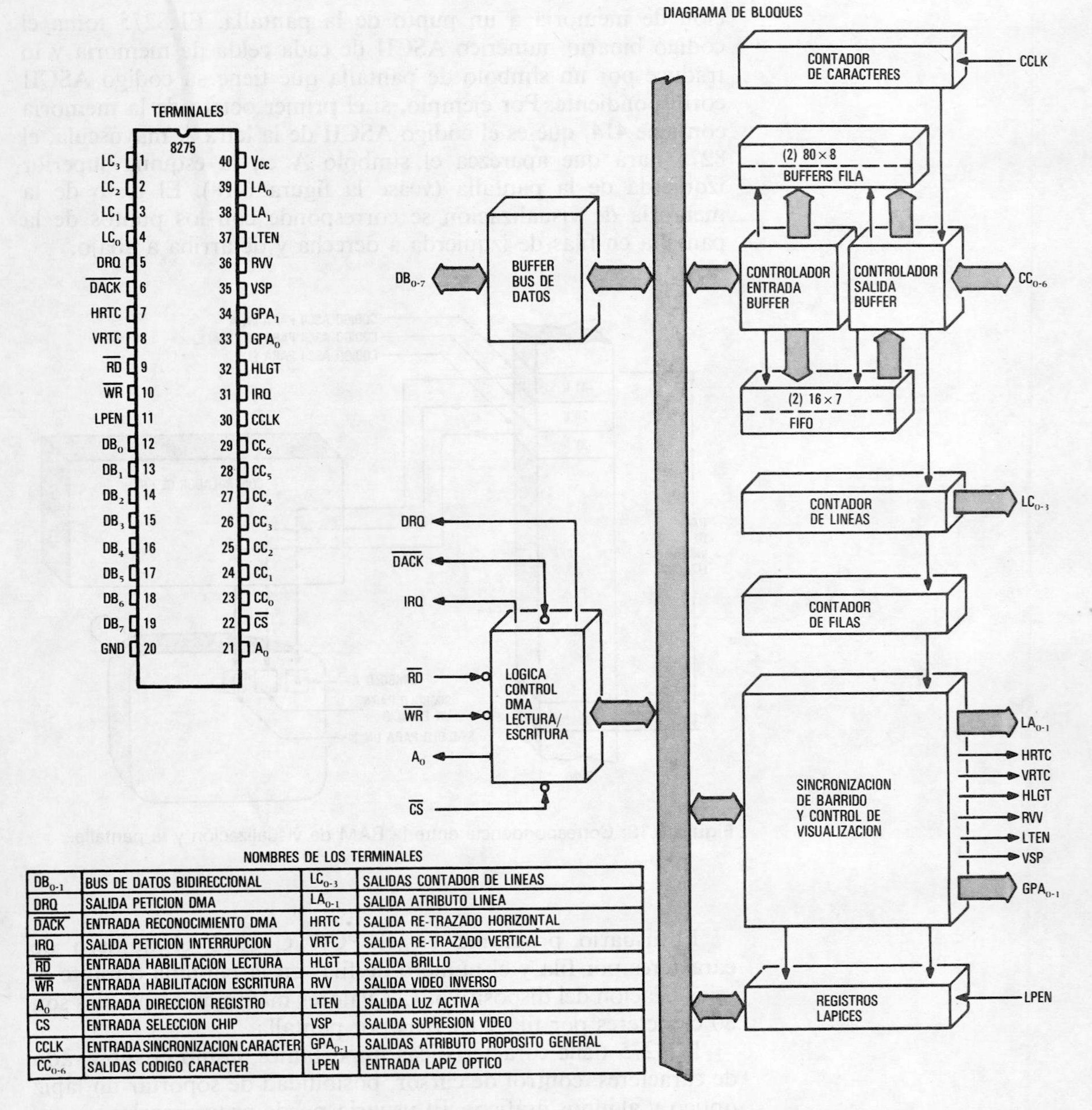
La rutina del terminal comienza llamando a las rutinas de inicialización del PPIC 8255 y del PSIC 8251. Ambas rutinas pueden verse también en el ejemplo. A continuación comienza el bucle. En primer lugar se pregunta al PPIC por la entrada del teclado. Si hay una entrada del teclado, se envía el carácter a la computadora externa a través del PSIC, y el bucle vuelve a comenzar. Si no había entrada, se pregunta al PSIC si hay algún carácter de la computadora externa. Si la computadora externa tiene un carácter, se envía a la pantalla a través del PPIC.

Tres de las rutinas (CINP, XINP y XOUT) usan uno de los bits de estado del octeto de estado para determinar cuándo se debe realizar la transmisión. La rutina de salida para la pantalla utiliza un bucle de temporización para asegurarse que el dispositivo no recibe información más rápidamente de lo que puede gestionarla.

Intel tiene ahora un chip, el Receptor/Transmisor Asíncrono Universal Multifunción 8256 (MUART: Multifunction Universal Asynchronous Receiver/Transmitter), que combina la mayoría de las funciones del PSIC 8251, PPIC 8255 y el PIC 8259. Tiene un port serie (con 13 frecuencias en baudios incorporadas), dos ports paralelos (uno para datos y otros para el estado) y otros elementos interesantes, como temporizadores, y un controlador de interrupción.

El Controlador Programable de CRT 8275

El Controlador Programable de CRT 8275 (PCRTC: Programmable CRT Controller) es un chip de 40 terminales que proporciona la sincronización y el control de la visualización de texto sobre la pantalla. El texto se almacena en una sección de memoria reservada a este efecto y que recibe el nombre de RAM



DB_{0-7}	BUS DE DATOS BIDIRECCIONAL	LC_{0-3}	SALIDAS CONTADOR DE LINEAS
DRQ	SALIDA PETICION DMA	LA_{0-1}	SALIDA ATRIBUTO LINEA
$\overline{DACK}$	ENTRADA RECONOCIMIENTO DMA	HRTC	SALIDA RE-TRAZADO HORIZONTAL
IRQ	SALIDA PETICION INTERRUPCION	VRTC	SALIDA RE-TRAZADO VERTICAL
$\overline{RD}$	ENTRADA HABILITACION LECTURA	HLGT	SALIDA BRILLO
$\overline{WR}$	ENTRADA HABILITACION ESCRITURA	RVV	SALIDA VIDEO INVERSO
A_0	ENTRADA DIRECCION REGISTRO	LTEN	SALIDA LUZ ACTIVA
CS	ENTRADA SELECCION CHIP	VSP	SALIDA SUPRESION VIDEO
CCLK	ENTRADA SINCRONIZACION CARACTER	GPA_{0-1}	SALIDAS ATRIBUTO PROPOSITO GENERAL
CC_{0-6}	SALIDAS CODIGO CARACTER	LPEN	ENTRADA LAPIZ OPTICO

Figura 6.17: Controlador Programable de CRT. Terminales y estructura interna.

de visualización de texto. La CPU coloca el código ASCII *digital* de los caracteres del texto en esta RAM de visualización, y el PCRTC genera las señales de *video* necesarias para visualizar los símbolos correspondientes sobre la pantalla.

El PCRTC 8275 barre repetidamente (aproximadamente unas 60 veces por segundo) la memoria de visualización de la computadora, comenzando por una posición específica, y durante un número de octetos específico, haciendo corresponder cada posi-

ción de memoria a un punto de la pantalla. El 8275 toma el código binario numérico ASCII de cada celda de memoria y lo traduce por un símbolo de pantalla que tiene su código ASCII correspondiente. Por ejemplo, si el primer octeto de la memoria contiene 414, que es el código ASCII de la letra A mayúscula, el 8275 hará que aparezca el símbolo A en la esquina superior izquierda de la pantalla (véase la figura 6.18). El resto de la memoria de visualización se corresponde con los puntos de la pantalla en filas de izquierda a derecha y de arriba a abajo.

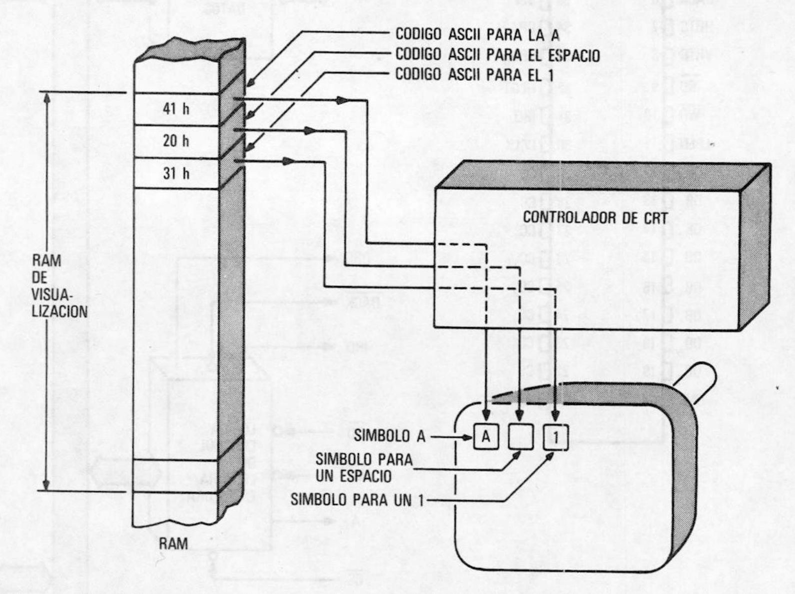


Figura 6.18: Correspondencia entre la RAM de visualización y la pantalla.

El usuario programa en el PCRTC 8275 el número de caracteres por fila y el número de filas en la pantalla durante la inicialización del dispositivo. Los valores máximos aceptables son 80 caracteres por fila y 64 filas por pantalla.

El 8275 tiene características importantes, como el subrayado de caracteres, control de cursor, posibilidad de soportar un lápiz óptico y algunos gráficos. El usuario puede programarlas vía un port de control especial en el espacio de E/S.

Muchas computadoras personales y terminales tienen una placa completa con la lógica de control de la visualización del texto. Con un chip controlador de CRT se disminuye significativamente el coste, el espacio y el consumo de potencia de dicha circuitería, a costa de una flexibilidad reducida.

El Controlador de Discos Flexibles de Simple/Doble Densidad 8272

El Controlador de Discos Flexibles de Simple/Doble Densidad 8272 (FDC: Floppy Disk Controller) (véase la figura 6.19) es un chip de 40 terminales, que provee la circuitería de control para la interfaz de hasta cuatro discos flexibles con una computadora

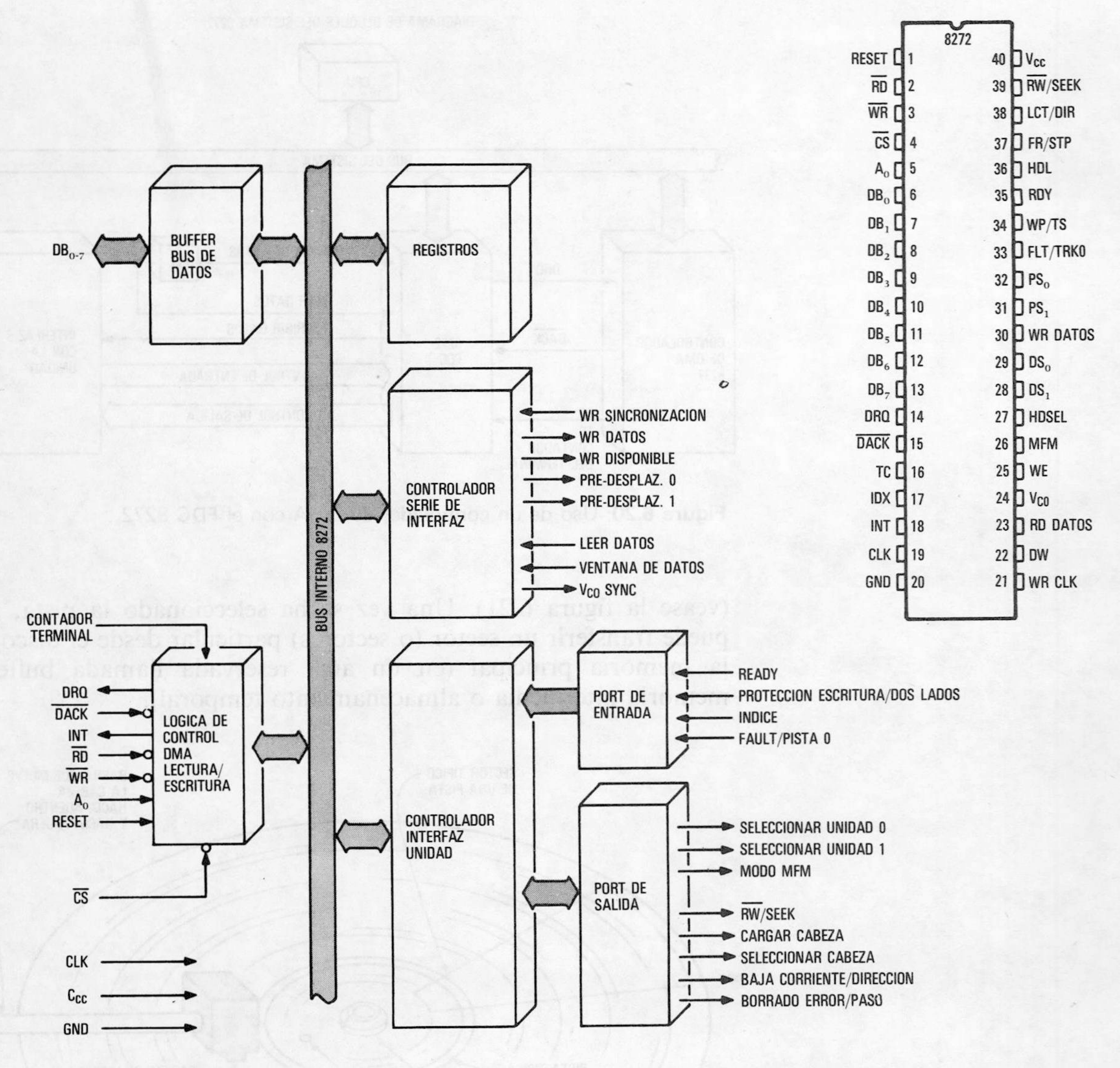


Figura 6.19: Controlador de Discos Flexibles 8272.

basada en el 8080, 8085, 8086, 8088 y la mayoría de microprocesadores. Es compatible con los formatos de discos de IBM de simple (IBM 3740) y doble densidad (IBM Sistema 34).

El FDC 8272 puede trabajar solo o junto a un controlador de DMA (véase la figura 6.20).

Los datos se organizan sobre los discos flexibles en círculos concéntricos llamados *pistas,* y cada pista se divide en cierto número de sectores. Hay una cabeza lectora/escritora que viaja por encima (o por debajo) del disco, girando a unas 5 revoluciones por segundo. Las pistas se seleccionan moviendo «paso a paso» la cabeza de lectura/escritura a lo largo de un eje radial

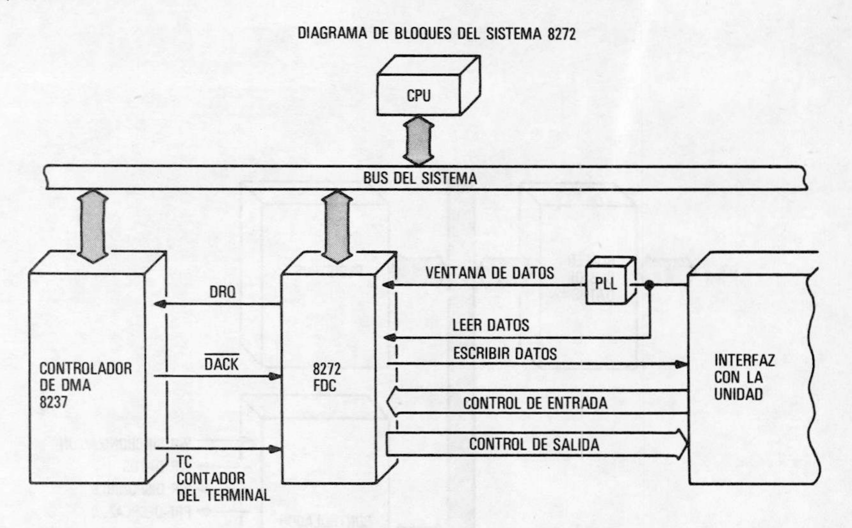


Figura 6.20: Uso de un controlador de DMA con el FDC 8272.

(véase la figura 6.21). Una vez se ha seleccionado la pista, se puede transferir un sector (o sectores) particular desde el disco a la memoria principal (en un área reservada llamada buffer, memoria intermedia o almacenamiento temporal).

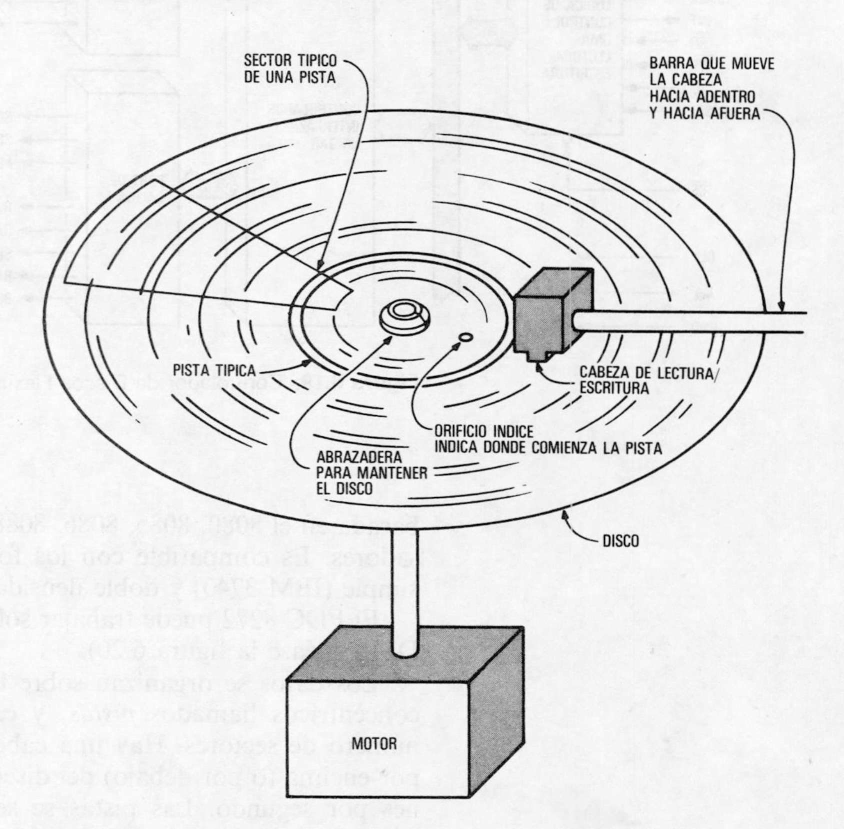


Figura 6.21: Estructura del disco flexible.

El FDC 8272 tiene disponibles 15 órdenes de control del disco básicas, incluyendo órdenes de movimiento de la cabeza de lectura/escritura de una unidad determinada a una cierta pista, de transferencia de sectores de datos entre la unidad seleccionada y la computadora, búsqueda de sectores dependiendo de determinados formatos, formato de discos, y de definición de algunos parámetros de sincronización. Cada orden produce tres etapas: 1) la orden en sí, 2) la acción y 3) el resultado. Durante la etapa de «orden», la CPU envía una serie de bits (la orden) al FDC. Durante la etapa de «acción», el FDC realiza una acción en el disco apropiado, normalmente transfiriendo datos a o desde el FDC y la CPU (o la memoria del sistema, en el caso de una transferencia DMA). Durante la etapa de «resultado», la CPU recibe una serie de octetos (el resultado) del FDC.

Más concretamente, las órdenes son de la siguiente forma:

Leer datos (Read data). Esta orden hace que se transfieran uno o más sectores de datos desde el disco a la computadora. Si se usa un controlador de DMA, se pueden colocar los octetos de los sectores definidos directamente en la memoria central, y en caso contrario, se pueden llevar los octetos a través de todo el procesador. Esta es una situación en la cual se puede utilizar muy eficientemente un IOP 8089. Hay en realidad 9 octetos de órdenes que definen la orden de lectura, conteniendo información sobre el número de la unidad de discos flexibles, lado en que se lee, número de sector, número de octetos por sector y longitud del sector. El número de pista no se incluye ya que la selección de pista lo realiza otra orden (la búsqueda).

Leer identificador. Esta orden hace que el FDC lea la identificación del sector siguiente que va a colocarse bajo la cabeza lectora/escritora del disco seleccionado. Esta información se manda a la CPU como parte de los resultados.

Leer dato borrado. Esta orden es muy similar a la de leer datos, excepto en que hace que se lea un sector incluso si hay una marca especial electrónica al comienzo del sector (llamada la marca de dirección de dato), que indica que el sector se ha borrado (qué está fuera de servicio).

Leer una pista. Esta orden lee una pista de la unidad de discos, y la envía a la CPU (a la memoria principal si es una transferencia DMA).

Busca por igual, por mayor o igual, o por menor o igual. Estas órdenes de búsqueda se utilizan para detectar formatos especiales en el disco, cosa que puede ser útil para distintos propósitos con grandes bases de datos. Por ejemplo, se pueden usar estas órdenes para buscar una cierta clave indicando un cierto cliente.

Especificar. Esta orden permite dar valores a los parámetros de sincronización HUT (Head Unload Time), HLT (Head Load

Time) y SRT (Step Rate Time). Al ser programables estos parámetros, se pueden adaptar diversas unidades de disco a este chip.

Escribir dato. Esta orden transfiere uno o más sectores de datos de la computadora a la unidad de discos seleccionada.

Formatear una pista. Antes de que se pueda utilizar un disco debe formatearse. Esta orden se encarga de hacerlo automáticamente, para un sector dado. Para formatear el disco completo, se debe entrar en un bucle que llame repetidamente a esta orden.

Escribir un dato borrado. Funciona igual que la orden de escritura de datos, con la excepción de que coloca una señal indicando «borrado» en la marca de dirección de dato para ese selector. Así la información queda allí, pero el sector oficialmente está fuera de servicio.

Buscar. Esta orden hace que la cabeza de lectura/escritura se posicione en la pista especificada por el usuario.

Volver a la pista 0. Como su nombre indica, esta orden recalibra la unidad de discos, llevando la cabeza de lectura/escritura a la pista 0.

Preguntar por el estado de las interrupciones. EL FDC puede interrumpir a la CPU por varias razones. Esta orden hace que el FDC devuelva un par de octetos, uno de los cuales indica la causa de la última interrupción ocurrida. Esta orden es necesaria después de las de buscar o volver a pista 0, para obtener información sobre el estado.

Preguntar por el estado de la unidad. Con esta orden se puede obtener un octeto del Controlador 8272 con información sobre la unidad de discos en servicio, si hay o no error, si el disco está o no protegido para la escritura, y si la unidad de discos está o no ocupada.

Como muchos controladores de interfaz, el FDC 8272 contiene dos registros accesibles, un registro de estado y uno de datos. Al registro de estado se puede acceder en cualquier instante. Sus 8 bits indican si el controlador y las unidades individuales están ocupadas o preparadas, y cosas como la dirección del flujo de datos (del procesador al FDC o viceversa). El registro de datos es en realidad un port bidireccional o una pila de registros en la cual se guardan órdenes, parámetros, el estado y los datos. Para introducir información en esta pila (durante la etapa de «orden»), u obtener información de ella (durante la etapa de «resultados») se sigue un método de sondeo. Es decir, antes de que transfiera cada octeto, se debe preguntar por el estado (dentro de un bucle software), hasta que el FDC esté preparado para realizar la transferencia.

Al igual que en el caso de la circuitería de visualización de textos, muchas computadoras incluyen una placa completa con cierto número de chips de propósito general para gestionar lo que este chip de Intel que estamos estudiando hace él solo. El 8272, en consecuencia, ahorra espacio, baja los costes de diseño e implementación, consume menos y aumenta la manejabilidad de la computadora.

CONCLUSIONES

En los últimos capítulos hemos estudiado los chips procesadores de 16 bits de Intel: las CPU 8086 y 8088, el chip IOP 8089 y el NDP 8087. En este capítulo se han visto los chips adicionales necesarios para la construcción de una computadora completa. Ahora estamos en condiciones de volver al capítulo 2, y entender mucho mejor los dos modelos de computadoras, los chips que los componen y sus conexiones.

Los chips de este capítulo caen dentro de cinco categorías: 1) de sincronización, 2) de interfaz con el bus, 3) controladores de sistema, 4) controladores de dispositivos de propósito general y 5) controladores de dispositivos de propósito especial.

Cada chip de los estudiados se diseñó para liberar al procesador del control directo de ciertas tareas, sea la CPU, o sea un procesador periférico como el IOP 8089. Cada tarea *lógica* se asigna a un chip *físico* diferente, haciendo al sistema más fácilmente entendible. Este es un ejemplo de la tendencia actual de diseño modular que hace que la computadora sea no sólo más fácil de diseñar, sino también más fácil de arreglar en caso de mal funcionamiento.

Capítulo 7

Once programas-ejemplo para el 8086/8088

En este capítulo presentaremos una serie de programas ejemplo, comenzando por casos muy sencillos y avanzando hacia ejemplos más importantes, de complejidad moderada. El lector puede, con un mínimo de dificultad, intentar ejecutar estos programas en su propio sistema. Si es así, dicho sistema necesita indispensablemente ciertos elementos hardware: una Placa Procesadora Dual Godbout 8085/8088, y unas unidades de discos flexibles, con el sistema operativo CP/M-80. Si además posee una visualización de texto, y una RM de visualización, se podrán ver claramente los resultados del primer par de programas. Por otro lado, si su sistema no es tan bienaventurado, podemos ofrecerle una forma de «echar un vistazo» a los resultados sobre una pantalla ficticia. De hecho, no es absolutamente necesario que el sistema funcione bajo CP/M-80. Las primeras versiones de estos programas se desarrollaron en realidad bajo un sistema operativo distinto a CP/M. Los ejemplos que presentamos no pretenden ser de un exagerado virtuosismo, sino que intentan tan sólo ser unos ejemplos útiles y naturales, de una complejidad mínima al principio y moderada al final, justo del tipo de programas que cualquiera que quiera mejorar la eficiencia de su sistema va a tener que escribir.

En este capítulo, el juego de instrucciones del 8086/8088 se ha dividido de forma distinta que el capítulo 3. Las instrucciones se han clasificado en grupos de una manera mucho más didáctica, comenzando por las de transferencia de datos, pasando a continuación a las instrucciones de control de bucles y, finalmente, a los conceptos aritméticos más avanzados.

Daremos instrucciones paso-a-paso de cómo ensamblar, encadenar, cargar y ejecutar los programas-ejemplo. La teoría general de cómo desarrollar programas en lenguaje ensamblador ya se vio en el capítulo 2. Supondremos que el lector está lo suficientemente familiarizado con la terminología que allí se introdujo.

PREPARACION

Los programas-ejemplo se ejecutan en una CPU 8088 con la Placa Procesadora Dual Godbout 85/88. Preparamos su sistema basado en el bus S-100 en un modo experimental. Si el lector quiere preparar su sistema de una forma similar e intentar ejecutar cada uno de los ejemplos, necesitará el siguiente hardware: un sistema de discos flexibles, un sistema de visualización de textos por medio de una RAM de visualización (opcional), y una

placa procesadora Dual Godbout 85/88. En cuanto al software, necesitará un sistema operativo CPM-80 configurado según su sistema, y un ensamblador-cruzado de Microsoft para el 8086/8088 (XMACRO-86). La figura 7.1 muestra las necesidades de hardware, y cómo preparar la placa Godbout para nuestros propósitos. En particular, debe asegurarse de que los interruptores estén de manera que el 8088 haga un *arranque en frío* cada vez.

1) Bus S.100.

2) Sistema de discos flexibles con sistema operativo CP/M.

3) Placa de visualización de texto, con RAM de visualización. Si es posible, colocarla de forma que la RAM comienze en la E800H.

4) Teclado o terminal de entrada vía un port de E/S.

5) Placa procesadora dual Godbout 85/88. Preparar los interruptores de la siguiente manera:

   a) SW1 posición 4, *off*. Para que el 8085 no re-inicialize cuando se pase control del 88 al 85.

   b) SW1, posición 5, *on*. Para que el 8088 no re-inicialice cuando se pase control del 85 al 88.

   c) Otras posiciones SW1, tal como los necesite su sistema. Vea el User's Manual.

   d) SW2 (alimentación sobre dirección salto), tal como lo necesite el sistema.

   e) SW3, preparado para la dirección de port «estándar», la FDH.

Figura 7.1: Necesidades hardware para la ejecución de los programas-ejemplo.

Una vez tenga colocado el hardware, será necesario entrar (por teclado) un programa cargador escrito en código del 8085 (véase la figura 7.2). Utilice como código fuente el fichero L88.ASM. Este fichero se debería ensamblar con el ensamblador ASM (que viene con CP/M) y convertir a fichero COM con el programa LOAD del CP/M. Con esto se convertiría en una orden del sistema operativo CP/M con el nombre de L88. El programa L88 es un *inicializador* más que un *cargador*, ya que cuando se le llama, da opciones para, o bien volver a CP/M, saltando a un programa del 8088, o visualizar la pantalla (que puede ser ficticia) sobre el dispositivo de salida de consola.

Antes de ensamblar el L88-ASM seguramente tendrá que modificar el *igualador* (equate) de su sistema en particular:

```
VIDEO   EQU   0E800H
```

Esta es la dirección real de memoria de comienzo de placa de RAM de visualización de textos *(pantalla video)*. Si no posee esta placa, puede reservar un área de RAM sin ocupar, preferiblemen-

te por encima de la posición 2000H (vamos a cargar nuestro software de la posición 100H a la 2000H). Se necesitan menos de 2 octetos de RAM para esta pantalla. Verá sentencias EQU similares en varios de los programas ejemplo. Como las EQU de los ejemplos son para un 8086 con las *direcciones segmentadas,* debería prepararla a 1/16 del valor usado en el programa cargador (L88). Por ejemplo, si en el libro utilizamos E800H en el programa cargador, utilizaremos E80H en los programas ejemplo del 8088, y si usamos 2000H en el cargador, deberemos poner 200H en los programas ejemplo.

La opción «ir a 8088» del cargador permite especificar, opcionalmente, una dirección hexadecimal del 8088 de la forma:

Segmento: desplazamiento

El valor por defecto es 100:0. Esto significa que la posición real de la memoria por defecto será:

1000H = 100H * 10H + 0H

Cuando se ejecuta la orden «ir a 8088», el programa cargador carga unos cuantos octetos de código 8088 y pasa control al 8088. El 8088 comienza a ejecutar este código que a su vez salta a la dirección especificada. El código del 8088 se posiciona a partir de la dirección FFFF0H, que es desde donde el 8088 hace su arranque en frío. Si tiene las placas antiguas S100 estándar de 16 bits, la posición de comienzo debe sustituirse por la FFF0H. Es decir, el arranque de direccionamiento de 16 bits sobre la placa de memoria de 16 bits simplemente ignora los 4 bits superiores de los 20 bits de dirección. En cualquier caso se deben reservar unos pocos octetos (13) de memoria RAM libre, en las posiciones superiores de memoria.

Es interesante resaltar que una de las formas de pasar de la CPU 8085 a la CPU 8088 consiste en ejecutar una instrucción IN en una dirección de un port previamente puesto a 1 (la OFDH en este caso). Análogamente, el paso del 8088 al 8085 puede hacerse con la instrucción INB (IN octeto).

```
                ; CARGADOR PARA EL 8088
                ;
                ; Este programa activa al 8088 desde el CP/M
                ;
0005 =          BDOS     EQU   0005H     ; servicio CP/M
0000 =          WARM     EQU   0000H     ; comienzo en caliente
000D =          CR       EQU   0DH       ; <CR>
000A =          LF       EQU   0AH       ; <LF>
0020 =          SPACE    EQU   20H       ; espacio
FFF9 =          OFFSET   EQU   0FFF9H    ; desplazamiento
FFFB =          SEG      EQU   0FFFBH    ; dirección del segmento
0100 =          CODE     EQU   0100H     ; segmento código por omisión del 8088
0040 =          STACK    EQU   0040H     ; segmento de pila del 8088
0000 =          ENTER    EQU   0000H     ; punto de entrada por omisión del 8088
E800 =          VIDEO    EQU   0E800H    ; dirección RAM de visualización
```

```
003C =          LINELEN   EQU   60          ; longitud línea video
0005 =          LINENUM   EQU   5           ; número líneas video
                ;
0100                      ORG   100H
                ;
0100 210402     OPT:      LXI   H,MESS1     ; se visualiza la cabecera
0103 CD3E01               CALL  MESOUT      ; de la tabla de opciones
                LOOP:
0106 211A02               LXI   H,MESS2     ; se visualiza una explicación
0109 CD3E01               CALL  MESOUT      ; dé la opción
010C CDEB01               CALL  CONIN       ; entrar un carácter
010F FE45                 CPI   'E'         ; si es E, entonces
0111 CA0000               JZ    WARM        ; saltar al CP/M, arranque en caliente
0114 FE47                 CPI   'G'         ; si es G, entonces
0116 CA2A01               JZ    J8088       ; saltar al 8088
0119 FE44                 CPI   'D'         ; si es D, entonces
011B CAA401               JZ    DISP        ; visualizar pantalla
011E C32101               JMP   ERROR       ; en otro caso, error
                ;
0121 21B802     ERROR:    LXI   H,MESS4     ; se visualiza un
0124 CD3E01               CALL  MESOUT      ; mensaje de error
0127 C30601               JMP   LOOP        ; y se vuelve a intentar
                ;
012A CD4801     J8088:    CALL  LOAD88      ; cargar el salto por omisión al 8088
012D CD5901               CALL  GETADD      ; obtener la dirección de salto
0130 DA2101               JC    ERROR
0133 DBFD                 IN    0FDH        ; entra en funciones el 8088
                ;
0135 21A102               LXI   H,MESS3     ; se visualiza la
0138 CD3E01               CALL  MESOUT      ; vuelta del 8088
013B C30001               JMP   OPT         ; ¿se vuelve a intentar?
                ;
                MESOUT:                     ; rutina de visualización del mensaje
013E 7E                   MOV   A,M         ; obtener carácter
013F 23                   INX   H           ; carácter siguiente
0140 B7                   ORA   A           ; ¿era el final?
0141 C8                   RZ                ; si lo era, volver
0142 CDDE01               CALL  CONDUIT     ; sacar el carácter
0145 C33E01               JMP   MESOUT      ; realizar un bucle
                ;
                LOAD88:                     ; carga la rutina de código de salto al 8088
0148 11F701               LXI   D,JMP88     ; apunta a la copia
014B 21F0FF               LXI   H,0FFF0H    ; apunta al destino
014E 0E0D                 MVI   C,LENGTH    ; muchos octetos
0150 1A         L88LP:    LDAX  D           ; obtener un octeto
0151 13                   INX   D           ; octeto fuente siguiente
0152 77                   MOV   M,A         ; pon el octeto
0153 23                   INX   H           ; siguiente octeto de destino
0154 0D                   DCR   C           ; decrementar contador
0155 C25001               JNZ   L88LP       ; y buclar
0158 C9                   RET               ; vuelta
                ;
0159 CDEB01     GETADD:   CALL  CONIN       ; segmento primero
015C FE0D                 CPI   CR          ; ¿vuelta de carro?
015E C8                   RZ                ; utilizar omisión en ambos
015F FE3A                 CPI   ':'         ; ¿dos puntos?
0161 CA6F01               JZ    GETAD1      ; utilizar omisión en seg.
0164 CD7F01               CALL  HEX16       ; obtener 16 bits de hex
0167 78                   MOV   A,B
0168 FE3A                 CPI   ':'         ; buscar dos puntos
016A 37                   STC               ; poner a 1 el acarreo para error
016B C0                   RNZ               ; sino, error
016C 22FBFF               SHLD  SEG         ; guardar la dirección del segmento
016F CDEB01     GETAD1:   CALL  CONIN       ; ¿comienza el número siguiente?
0172 CD7F01               CALL  HEX16       ; obtener 16 bits de hex
0175 78                   MOV   A,B
0176 FE0D                 CPI   CR          ; verificar vuelta de carro
0178 37                   STC               ; poner a 1 el acarreo para error
0179 C0                   RNZ               ; volver si hay error
017A 22F9FF               SHLD  OFFSET      ; guardar desplazamiento
017D B7                   ORA   A           ; borrar acarreo
017E C9                   RET
                ;
017F 210000     HEX16:    LXI   H,0         ; comienzo limpio
0182 CD9301     HEXLP:    CALL  CONDIG      ; dígito limpio
0185 D8                   RC                ; volver si hay error
0186 29                   DAD   H           ; x2
0187 29                   DAD   H           ; x4
0188 29                   DAD   H           ; x8
```

```
0189 29                  DAD    H             ; x16
018A 85                  ADD    L             ; sumar el digito
018B 6F                  MOV    L,A
018C CDEB01              CALL   CONIN         ; siguiente carácter
018F 47                  MOV    B,A           ; salvaguardarlo
0190 C38201              JMP    HEXLP         ; y realizar un bucle
                ;
0193 D630       CONDIG:  SUI    'Ø'           ; ¿por debajo de cero?
0195 D8                  RC                   ; si es cierto, volver
0196 FEØA                CPI    1Ø            ; ¿por encima de 9?
0198 3F                  CMC
0199 DØ                  RNC                  ; si no es cierto, volver
019A D611                SUI    'A'-'Ø'       ; ¿por debajo de A?
019C D8                  RC                   ; si es cierto, volver
019D FE06                CPI    'F'-'A'+1     ; ¿por encima de F?
019F 3F                  CMC
01AØ D8                  RC                   ; si es cierto, error
01A1 C6ØA                ADI    1Ø            ; sumar 1Ø si va bien
01A3 C9                  RET
                ;
01A4 CDD301     DISP:    CALL   CCRLF
01A7 21C902              LXI    H,MESS5       ; margen
01AA CD3E01              CALL   MESOUT
01AD CDD301              CALL   CCRLF
01BØ 2100E8              LXI    H,VIDEO       ; RAM de visualización
01B3 1605                MVI    D,LINENUM     ; número de líneas
01B5 1E3C       DISP1:   MVI    E,LINELEN     ; longitud de la línea
01B7 7E         DISP2:   MOV    A,M           ; obtener octeto
01B8 23                  INX    H             ; apuntar al siguiente
01B9 CDDE01              CALL   CONOUT        ; sacarlo
01BC 1D                  DCR    E             ; ¿fin de línea?
01BD C2B701              JNZ    DISP2
01CØ CDD301              CALL   CCRLF         ; acabar
01C3 15                  DCR    D             ¿fin de pantalla?
01C4 C2B501              JNZ    DISP1
01C7 21C902              LXI    H,MESS5       ; mostrar margen
01CA CD3E01              CALL   MESOUT
01CD CDD301              CALL   CCRLF
01DØ C30601              JMP    LOOP
                ;
01D3 3EØD       CCRLF:   MVI    A,CR          ; vuelta de carro
01D5 CDDEØ1              CALL   CONOUT
01D8 3EØA                MVI    A,LF          ; nueva línea
01DA CDDEØ1              CALL   CONOUT
01DD C9                  RET
                ;
                CONOUT:                       ; rutina de salida por consola
01DE E5                  PUSH   H             ; salvaguardar registros
01DF D5                  PUSH   D
01EØ C5                  PUSH   B
01E1 5F                  MOV    E,A           ; E tiene el carácter
01E2 ØEØ2                MVI    C,2           ; C tiene la función
01E4 CDØ5ØØ              CALL   BDOS          ; llamar a CP/M
01E7 C1                  POP    B             ; restaurar registros
01E8 D1                  POP    D
01E9 E1                  POP    H
01EA C9                  RET                  ; y volver
                ;
01EB E5         CONIN:   PUSH   H             ; salvaguardar registros
01EC D5                  PUSH   D
01ED C5                  PUSH   B
01EE ØEØ1                MVI    C,1           ; C tiene la función
01FØ CDØ5ØØ              CALL   BDOS          ; llamar al CP/M
01F3 C1                  POP    B             ; restaurar registros
01F4 D1                  POP    D
01F5 E1                  POP    H
01F6 C9                  RET                  ; y volver
                ;
                JMP88:                        ; código salto 8088
01F7 B8                  DB     ØB8H          ; MOVI AX,STACK
01F8 4000                DW     STACK
01FA 8EDØ                DB     Ø8EH,ØDØH     ; MOV  SS,AX
01FC BCØØ01              DB     ØBCH,ØØØH,ØØ1H ; MOVI SP,1ØØH
01FF EA                  DB     ØEAH          ; JMP  ENTER,CODE
0200 0000                DW     ENTER
0200 0001                DW     CODE
000D =          LENGTH   EQU    $-JMP88
                ;
                ; message table
```

```
0204 3830383820   MESS1:   DB  '8088 Loader Prosram',CR,LF,0
021A 5479706520   MESS2:   DB  'Type your command:',CR,LF
022E 2020204520            DB  '   E                 Exit to CP/M',CR,LF
0253 202020475B            DB  '   G[ses:offset]<CR> Go to 8088',CR,LF
0276 2020204420            DB  '   D                 Display screen',CR,LF
029D 0D0A2A00              DB  CR,LF,'*',0
02A1 0D0A526574   MESS3:   CB  CR,LF,'Returned from 8088', CR, LF, 0
02B8 0D0A53796E   MESS4:   DB  CR,LF,'Syntax error',CR,LF,0
02C9 2A2A2A2A2A   MESS5:   DB  '******** memory ********',0
                  ;
02E2                       END
```

Figura 7.2: Programa cargador.

El resto de los programas se han escrito en código 8086/8088. Todos necesitan el programa cargador anterior.

PROGRAMAS

Para cada grupo de instrucciones hemos escrito un programa-ejemplo separado, que utiliza sólo instrucciones de ese grupo o de otro anterior. Hemos preparado los ejemplos para mostrar la potencia del sistema de segmentación Intel para empaquetar módulos de código. Colocamos cada uno de los conjuntos principales de subrutinas en un segmento aparte, para posteriormente poder tratar cada segmento como una unidad lógica y poder moverlos en memoria y ejecutarlos correctamente cambiando los contenidos de los registros de segmentación.

En cada caso se lista primero el nuevo grupo de instrucciones y después se explica qué hace el programa y cómo trabaja. Nunca utilizaremos todas las instrucciones del grupo en el ejemplo, porque llevaría a programas demasiado forzados y artificiales. A cambio, usaremos más y más instrucciones de cada grupo en los ejemplos posteriores.

Instrucciones de transferencia de datos

El primer grupo que estudiaremos engloba a las instrucciones de *transferencia de datos*. Dichas instrucciones mueven datos de una parte a otra del sistema; de y a la memoria principal, de y a los registros de datos, ports de E/S, y registros de segmentación.

Las instrucciones de *transferencias de datos* son las siguientes:

MOV	Transfiere
XCHG	Intercambia
IN	IN
OUT	OUT
XLAT	Traduce
LEA	Carga la dirección efectiva
LDS	Carga el segmento de datos
LES	Carga el segmento extra
LAHF	Carga los indicadores en AH
SAHF	Guarda AH en los indicadores

Veamos un ejemplo para este grupo:

```
                          ; Ejemplo 1: (Transferencia de datos)
                          ;
                          ; Visualizar "HI" sobre la pantalla
                          ;
0000'                              ASEG
0E80                      VIDEO    EQU    0E80H
0000                      SCREEN   EQU    0000H

0000    BB 0E80           ENTER:   MOVI   AX,VIDEO        ; el segmento de datos es
0003    8E D8                      MOV    DS,AX           ; la RAM de visualización
                          ;
0005    C6 06 0000 48              MOVBI  SCREEN+0,'H'    ; 'H' en el extremo superior izquierdo
000A    C6 06 0001 49              MOVBI  SCREEN+1,'I'    ; 'I' a continuación
                          ;
000F    E4 FD                      INB    0FDH            ; vuelta al 8085
                          ;
                                   END                    ; Fin del segmento código
```

Figura 7.3: Ejemplo 1: Visualizar «HI» sobre la pantalla.

Comentarios: Este es un ejemplo bonito. Muestra cómo con sólo unas pocas instrucciones de transferencia de datos y algunos de los modos de direccionamiento se puede crear un efecto visible, en este caso, visualiza el mensaje «HI» por pantalla. Tal vez después de ver cómo trabaja, el lector pueda estar interesado en tratar de enviar un mensaje más largo, como «HI THERE». El programa constituye además un buen test de funcionamiento de su sistema con la CPU 8088.

El programa está en un segmento que puede colocarse en cualquier área de RAM libre. Nosotros lo cargaremos en la 1000H que será el «punto central» de memoria de estos programas-ejemplo. Esta posición es la de salto por omisión del programa cargador.

Para ejecutar este programa, utilice un editor para introducir el código fuente tal como aparece en la figura 7.3 (excepto «el igualador de video«). Guárdelo en un fichero en disco con el nombre de E1.MAC (la extensión .MAC será necesaria para poder realizar correctamente el siguiente paso). Ahora ensámblelo (traduzca el código fuente a código máquina parcial) con el ensamblador cruzado 8086 de Microsoft con la orden siguiente:

XM86=E1/L

Esta orden hará que el código fuente se lea del fichero E1.MAC y que el código objeto se guarde en el E1.REL. La /L fuerza a que se almacene un listado en el fichero E1.RRN. La orden anterior en una versión abreviada de la siguiente:

XM86 fichero-objeto, fichero-listado=fichero-fuente.

«fichero-objeto» es el nombre del fichero que contiene el código objeto; «fichero-listado» es el nombre de su fichero listado, y «fichero-fuente» es el nombre del fichero que contiene el código fuente en lenguaje ensamblador. Es mejor no incluir los nombres

de las extensiones porque el ensamblador ya las supone, y si por error colocara una extensión cambiada de sitio ¡puede incluso llegar a destrozar el código fuente original! Sin las extensiones, el ensamblador buscará un fichero código fuente con el nombre especificado y una extensión .REL, y producirá un fichero listado con el mismo nombre pero con la extensión .PRN. Normalmente se usa el mismo nombre principal en los tres ficheros de manera que la forma abreviada se guarda en las teclas.

El código objeto del E1.REL todavía no puede ejecutarse directamente. Antes necesita pasar por el editor de enlaces (véase capítulo 2) con la orden:

L80 E1,E1/N/E

Esta orden hace que se lea el código objeto del fichero E1.REL y se guarde en el fichero E1.COM. Antes de acabar, el programa editor de enlaces (encadenador) le dirá que es peligroso cargar el código en el punto indicado, pero permite que se haga. Se debe seleccionar la opción N (de «No»).

En este caso no se admiten abreviaturas. La sintaxis es:

L80 fichero-objeto-1, fichero-objeto-2,..., fichero-objeto-n, fichero-de carga/N, opciones

Aquí el fichero del código objeto:

Los ficheros-objeto-1, fichero-objeto-2,..., fichero-objeto-n

son ficheros fuente para la operación de edición de enlaces. Todos ellos se combinan en un único programa lenguaje-máquina. En nuestro caso hay un solo fichero objeto llamado E1.REL. A continuación viene «fichero-de-carga/N». La «/N» es en realidad un *conmutador de opción* que hace que el fichero anterior se defina como fichero de carga. En nuestro caso queremos que el fichero de carga tenga el nombre E1, por lo que ponemos «E1/N». Por último están las opciones. En nuestro caso hemos escogido la opción «/E», que hace que el editor de enlaces salvaguarde el fichero de carga (bajo el nombre especificado por la opción /N) y salga. Las extensiones de ficheros *no deberían* utilizarse en órdenes al editor de enlaces por la misma razón que no deberían usarse en las órdenes al ensamblador. Se *pueden* usar, pero se corre cierto riesgo. Si no se utilizan, el editor de enlaces busca los ficheros objeto con la extensión REL y produce un fichero de carga con la extensón COM.

Una vez se ha enlazado y conservado el programa, se carga en memoria con la orden:

DDT E1.COM

DDT quiere significar *herramienta de depuración dinámica* (Dynamic Debugging Tool), y es una orden al sistema operativo CP/M. (Para más información sobre la orden DDT y otras del CP/M, véase Mutha, Stephen; Waite, Mitchell. *CP/M Primer*. Indianápolis, Indiana: Howard W. Sams & Co., Inc.)

La siguiente orden DDT lleva al programa a su posición verdadera:

M100,2FF,1000

Si queremos colocarlo en algún otro sitio se debe cambiar el último número. Un control C a continuación hace que se vuelva al CP/M, y se debe introducir un:

L88

Con esto se empieza a ejecutar el programa cargador que hemos visto antes en este mismo capítulo. En la pantalla debe aparecer la palabra «HI». Si no tiene pantalla, teclee la opción *D* del cargador, y los caracteres de «visualización» sobre la pantalla ficticia. A continuación deberá volver el CP/M, colocar espacios (20H) en la pantalla (con la orden DDT de *rellenar),* re-cargar el ejemplo 1, y volver a comenzar el programa L88. Visualice la pantalla con la orden »D«, vaya al ejemplo 1 con la orden *G* y finalmente vuelva a visualizar la pantalla con otra *D*.

Ahora que ya sabemos cómo ejecutar este programa, veamos cómo trabaja.

El programa inicializa primero el segmento de datos de forma que apunte a la RAM de la pantalla de visualización. Con esto el programador puede pensar en la RAM de visualización como en un bloque de datos comenzando en la dirección cero. Notemos que el registro de segmentación se carga con E80H, que es 1/16 de la dirección RAM real. (Recuérdese que los contenidos de los registros de segmentación se multiplican por 16 cuando se calcula la dirección real.) Nótese asimismo que el preparar el registro de segmentación de datos necesita dos instrucciones. No se puede mover directamente una constante a un registro de segmentación. Si la dirección del segmento se guarda en memoria, entonces se puede llevar directamente al registro de segmentación. Aquí no hemos hecho esto debido a que dicho acceso a memoria necesitaría utilizar el segmento de datos, que es el registro que tratamos de inicializar. Podríamos, sin embargo, utilizar un prefijo anula-segmento y obtener la información del segmento de código (el segmento que contiene el programa).

La primera instrucción de este proceso de inicialización en dos pasos es una instrucción MOV con modo de direccionamiento inmediato, para la fuente y un modo registro para el destino. En las instrucciones del *doble-operando* de este procesador, el destino viene primero, luego una coma y a continuación la fuente[1].

En el ensamblador-cruzado Microsoft, el modo de direccionamiento inmediato se explicita con una *I* en el código simbólico de operación, como en esta primera instrucción. Sólo se usa el modo inmediato con el operando fuente, lo cual, pensándolo un poco, tiene sentido. Téngase en cuenta que, cuando una instrucción

---

[1] Esta ordenación particular de los operandos no es universal en todos los procesadores. Por ejemplo, en el lenguaje ensamblador de la minicomputadora PDP-11, la fuente siempre va primero.

tiene un operando en modo inmediato, el valor real del operando se guarda como parte del código máquina. Este dato es siempre la última parte del código máquina. Así, el operando se guarda dentro del flujo de instrucciones del procesador. En vista de cómo obtiene el 8086/8088 sus instrucciones, a través de una estructura tubular (pipeline), este es un mal sitio para colocar un operando destino, y un sitio muy bueno para colocar un operando fuente. Incluso en procesadores que no utilicen la estructura tubular, sigue siendo cierta. El tener el destino en el flujo de instrucciones es una mala práctica porque obliga a que el programa se modifique a sí mismo, y en general, un código *auto-modificable* es difícil de depurar y mantener.

Notemos que el destino de la primera instrucción, MOV, es un registro. De hecho es un registro muy especial: el acumulador AX. El 8088 trata esto como un caso especial y utiliza un código máquina más corto.

También hay que resaltar que la segunda instrucción es una transferencia de un registro a otro. El código máquina para la segunda instrucción es completamente distinto del de la primera aunque ambas son instrucciones MOV. Una instrucción MOV de registro a registro necesita sólo 2 octetos.

En el siguiente par de instrucciones el código ASCII de la H se lleva a la primera posición de la RAM de video, y el código ASCII para la *I* se coloca en la segunda posición. La placa de video hace que aparezcan los símbolos correspondientes en la fila superior de la pantalla, comenzando por el lado izquierdo. En estas instrucciones las fuentes están en modo inmediato, y el destino es una referencia directa a memoria. En las instrucciones que mueven la *H* y la *I* hay signos + en la expresión del destino. El ensamblador y no la CPU debe evaluar esta expresión para encontrar la dirección del destino (que es la posición siguiente a la RAM de visualización). Ambas instrucciones están en la modalidad de octeto, como puede verse por la *B* del código de operación simbólico. Esta modalidad resulta útil en el tratamiento de caracteres. Cuidado, la *B* aparece antes de la *I*.

Finalmente, la última instrucción hace que el 8088 devuelva control al 8085 a través de la instrucción IN de nuestro port estándar FDM. Es una instrucción de transferencia de datos que se utiliza con propósitos diferentes, el de pasar control de un procesador a otro.

Instrucciones simples de control de bucles

Las instrucciones *simples de control de bucles* posibilitan el tipo de control más básico de nuestros programas. Este grupo incluye las siguientes instrucciones:

INC	Incrementar
DEC	Decrementar
LOOP	Realizar un bucle
LOOPZ LOOPE	Realizar un bucle si cero (igual)
LOOPNZ LOOPNE	Realizar un bucle no cero (igual)
JCXZ	Salto si CX es cero

Casi cualquier programa útil tiene algún bucle. Un bucle es un bloque de código que se ejecuta varias veces. Hay cuatro tipos de bucles fundamentalmente: 1) *bucles sin fin,* 2) *bucles por conteo,* 3) *bucles «hasta»* y 4) *bucles «mientras».*

En el primer tipo, los bucles sin fin, el bloque de código se ejecuta repetidamente, sin que ningún control lo acabe. Es un tipo de bucle muy popular entre los principiantes. En BASIC tales bucles pueden pararse con un control *C* (la tecla del pánico). En FORTRAN o en lenguaje máquina, se debe recurrir a medidas más drásticas como desconectar la computadora o hacer un RESET. En BASIC, FORTRAN y Pascal, tales bucles se controlan con sentencias GOTO, aunque esta sentencia (y por tanto este tipo de bucle) esté mal vista en Pascal. En el lenguaje ensamblador del 8086/8088 se usa la instrucción JMP (salto al final del bucle). Para mantener el espíritu moderno del Pascal, también nosotros descartaremos este tipo de bucle, y no incluiremos la instrucción JMP en el grupo de instrucciones simples de control de bucles. Veamos un ejemplo:

```
START:                      ; comienzo del bucle
            ....            ; instrucciones
                            ; del bucle
            ....
            JMP     START   ; siempre vuelve a START
```

En el segundo tipo, los bucles por conteo, el bloque se repite un número determinado de veces *(el contador del bucle).* Estos bucles se controlan en BASIC mediante las sentencias FOR/NEXT, en FORTRAN mediante la sentencia DO, y en Pascal por la FOR. En el ensamblador 8086/8088 se debe cargar previamente el registro contador (CX) con el contador del bucle, y colocar la instrucción LOOP al final del bucle. Cada vez que se encuentra la instrucción LOOP el registro CX se decrementa automáticamente. Si el resultado no es cero, la instrucción LOOP salta al comienzo del bucle (especificado por el operando de la instrucción LOOP) y vuelve a comenzar. Así, por ejemplo:

```
            MOVI    CX,8    ; contador igual a 8
START:                      ; comienzo del bucle
            ....            ; instrucciones
                            ; del bucle
            ....
            LOOP    START   ; decrementar CX y
                            ; volver al comienzo del bucle
                            ; si CX no es cero
```

En el tercer tipo, los bucles hasta, se pregunta por una condición al final del bucle. Si la condición se verifica, el bucle termina (después de llegar al final). Hay *sentencias hasta* (UNTIL) en los lenguajes estructurados modernos como el Pascal y las versiones más nuevas de BASIC y FORTRAN. En el lenguaje ensamblador del 8086/8088, LOOPZ, LOOPE, LOOPNZ y LOOPNE se encargan de controlar este tipo de bucles. Realmen-

te, LOOPZ y LOOPE son nombres en lenguaje ensamblador distintos para la misma instrucción código máquina, así como LOOPNZ y LOOPNE. Combinan las virtudes del bucle por conteo con las del bucle hasta. Al igual que en los bucles por conteo, se puede cargar en el registro CX en contador del bucle y colocar la instrucción LOOP... al final del bucle. Si el indicador de cero está a 1 (para LOOPZ y LOOPE) o a 0 (para LOOPNZ y LOOPNE) al final del bucle (cuando se encuentra la instrucción LOOP...) el bucle continúa; en caso contrario acaba. Por ejemplo:

```
         MOVI    CX,100   ; contador a 100
START:                    ; comienzo del bucle
         ....             ; instrucciones
                          ; dentro del bucle
         ....
         CALL    TESTIT   ; esta subrutina
                          ; pone a 1 ZF su ya se ha hecho
         LOOPNZ           ; realiza un bucle hasta que se hace
```

En el cuarto tipo, los bucles mientras, se pregunta por una condición al principio del bucle. Si la condición *no* es cierta, el bucle termina. Como en la sentencia hasta, el bucle mientras es muy popular en los modernos lenguajes estructurados. En el ensamblador 8086/8088, lo más cercano a un bucle mientras es la instrucción JCXZ (Salto si CX es 0). Si se coloca esta instrucción al comienzo del bucle y si el contador del bucle es cero, el bucle termina. Esta instrucción proporciona un mejor control sobre los bucles por conteo. En los lenguajes de programación de alto nivel algunos bucles FOR funcionan de esta manera; es decir, no se ejecutan ni tan siquiera una vez si el contador de bucle es cero. Veamos un ejemplo de lenguaje ensamblador:

```
         MOVI    CX,50    ; contador a 50
START:   JCXZ    END      ; comienzo del bucle
                          ; contiene un test
         ....             ; instrucciones
                          ; dentro del bucle
         ....
         LOOP    START    ; decrementa y bucla
END:                      ; continúa el resto
                          ; del programa
```

Hemos incluido las instrucciones de incrementar (INC) y decrementar (DEC) en este grupo debido a que con ellas se puede contar o bien para control del bucle (cuando se usan junto al salto condicional) o alguna otra cantidad del bucle. Estudiaremos las instrucciones de salto condicional con el salto incondicional en otro grupo más adelante.

La figura 7.4 muestra un programa-ejemplo que utiliza instrucciones del grupo de control de bucles. El programa puede guardarse en un fichero E2.MAC. De nuevo, no se olvide de la instrucción EQU para video. El programa se puede ensamblar,

enlazar y ejecutar igual que en el primer ejemplo. Esto es, la orden:

XM86=E2/L

ensambla el fichero fuente generando un fichero código máquina reubicable (.REL) y un fichero listado (.PRN); y la orden:

L80 E2,E2/N/E

produce un fichero código máquina absoluto (.COM) con el nombre de E2.COM. Se necesita utilizar DDT para cargar y llevar el programa a su lugar, al igual que en el ejemplo anterior. Cuando utilice el «inicializador» para ejecutarlo, podría ver los resultados en la pantalla (ficticia, si es el caso).

```
                        ; Ejemplo 2: (Control simple de bucles)
                        ;
                        ; Visualizar el código ASCII sobre la pantalla
                        ;
0000'                           ASEG
0E80            VIDEO           EQU     0E80H
                        ;
0000    B8 0E80 ENTER:          MOVI    AX,VIDEO        ; apuntar al segmento de datos
0003    8E D8                   MOV     DS,AX           ; a la RAM de video
                        ;
0005    B0 20                   MOVBI   AL,20H          ; AL contiene el primer carácter a imprimir
                                                        ; de código ASCII
0007    BB 0000                 MOVI    BX,0            ; BX contiene la primera posición
000A    B9 0060                 MOVI    CX,96           ; CX contiene el contador
                        ;
000D    88 07   ASCII:          MOVB    [BX],AL         ; llevar el carácter a su lugar
000F    43                      INC     BX              ; posición siguiente
0010    40                      INC     AX              ; carácter siguiente
0011    E2 FA                   LOOP    ASCII           ; realizar un bucle
                        ;
0013    E4 FD                   INB     0FDH            ; vuelta al 8088
                                END
```

Figura 7.4: Ejemplo 2: Visualización del código ASCII sobre la pantalla.

Comentarios: Este ejemplo muestra cómo combinar las instrucciones de transferencia con las de control de bucles para visualizar eficazmente unos pocos caracteres más en la pantalla. En este caso se visualiza en la pantalla el código ASCII completo.

Este segmento se puede colocar en memoria en la posición 1000H para que cuadre con la dirección de salto por omisión del programa cargador.

Como en el ejemplo anterior, el registro de segmentación de los datos se inicializa en el primer par de instrucciones de forma que apunta a la RAM de video. Las siguientes instrucciones inicializan tres registros de datos antes de entrar en el bucle (los registros AL, BX y CX). La mitad másbaja del registro A contiene el código ASCII comenzando con 20H (porque no queremos *caracteres de control)*. El registro base (BX) apunta a la posición de memoria en la que queremos colocar los caracteres, cosa que se indica mediante el símbolo [BX] de modo de direccionamiento. Si inicializamos BX a 0, los caracteres se posicionarán en la RAM de visualización a partir de la primera

posición de memoria. El registro contador (CX) se utiliza para guardar el 96 (para nuestro subconjunto de código ASCII).

El bucle comienza con un rótulo, que le da nombre. En el programa se ha escogido el nombre ASCII.

Después se llevan el carácter a su lugar se incrementa a la vez el código ASCII y la posición, y se decrementa el contador. Notemos que hemos incrementado todo el AX, mientras que sólo AL contiene realmente el carácter. Esto se hace así para sacar partido de la forma optimizada del código de la instrucción INC de 1 octeto para incrementar registros de 16 bits. El contenido de AH no es importante y no cambia el resultado de la ejecución de este programa.

La última instrucción del bucle (LOOP) se encarga del control de éste. Decrementa automáticamente el contador, comprueba si el contador es cero, y hace un bucle si no lo es. Como la instrucción de control del bucle está al final de éste, el bucle siempre se ejecuta al menos una vez. Si pusiéramos inicialmente el contador a 0, el bucle se ejecutaría 64K-1 veces. Para asegurarse que no ocurre esto nunca, se puede poner la instrucción JCXZ al comienzo del bucle y saltar a donde se quiera si el contador es cero para comenzar.

El programa termina con una instrucción IN que devuelve control al 8085.

Instrucciones de test, comparación y saltos

Este grupo en realidad es una continuación del grupo de control de bucles. Incluye las siguientes instrucciones:

TEST		Verifica
CMP		Compara
JMP		Salta
JE	JZ	Salta si igual (cero)
JNE	JNZ	Salta si no igual (cero)
JS		Salta si signo (negativo)
JNS		Salta si no signo (no-negativo)
JP	JPE	Salta si paridad (par)
JNP	JOP	Salta si paridad (impar)
JO		Salta si hay capacidad excedida
JNO		Salta si no hay capacidad excedida
JB	JNAE	Salta si por abajo (no encima o igual)
JNB	JAE	Salta si no por abajo (encima o igual)
JBE	JNA	Salta si por abajo o igual (no encima)
JNBE	JA	Salta si no por abajo o igual (encima)
JL	JNGE	Salta si menor que (no mayor o igual)
JNL	JGE	Salta si no menor que (mayor o igual)
JLE	JNG	Salta si menor que o igual (no mayor)
JNLE	JG	Salta si no menor que o igual (mayor)

Aquí tenemos a la vez saltos condicionales e incondicionales así como instrucciones aritméticas y de *lógica virtual:* TEST y CMP. Por virtual se entiende que realizamos la operación (un AND lógico para TEST y una resta para CMP), actualizamos los indicadores en consonancia, pero no retenemos el resultado.

La figura 7.5 muestra el programa-ejemplo dedicado a las instrucciones de TEST, CMP y saltos. En este caso no se utiliza la pantalla de visualización, sino que en su lugar se establece comunicación a través de un port (serie o paralelo).

```
                Ejemplo 3: (TEST, CMP y Saltos)
                ;
                ; Programa de eco para la E/S
                ;
0000'                     ASEG
0086            IFSTAT    EQU     86H        ; port de estado de entrada
0083            IPDATA    EQU     83H        ; port de datos de entrada
0080            OPSTAT    EQU     80H        ; port de estado de salida
0082            OPDATA    EQU     82H        ; port de datos de salida
0008            IMASK     EQU     08H        ; máscara de estado de entrada
0004            OMASK     EQU     04H        ; máscara de estado de salida
0086            INITP     EQU     86H        ; port de inicialización
0014            INITM     EQU     14H        ; máscara de inicialización
                ;
0000    B0 14   CDINIT:   MOVBI   AL,INITM   ; procedimiento de
0002    E6 86             OUTB    INITP      ; inicialización
                ;
0004            LOOP:                        ; bucle principal
0004            CDIN:                        ; rutina de entrada
0004    E4 86             INB     IPSTAT     ; estado de la entrada
0006    A8 08             TESTBI  AL,IMASK   ; verificación del bit de estado
0008    75 FA             JNZ     CDIN       ; realizar un bucle hasta que esté preparado
000A    E4 83             INB     IPDATA     ; obtener el dato
000C    8A E0             MOVB    AH,AL      ; devolverlo en AH
                ;
000E    3C 03             CMPBI   AL,03      ; control C
0010    74 0C             JZ      RETURN
                ;
0012            CDOUT:                       ; rutina de salida
0012    E4 80             INB     OPSTAT     ; estado de la salida
0014    A8 04             TESTBI  AL,OMASK   ; verificación del bit de estado
0016    75 FA             JNZ     CDOUT      ; realizar un bucle hasta que esté preparado
0018    8A C4             MOVB    AL,AH      ; obtener el dato preparado
001A    E6 82             OUTB    OPDATA     ; sacarlo
                ;
001C    EB E6             JMPS    LOOP       ; siempre vuelve al comienzo del bucle
                ;
001E    E4 FD   RETURN:   INB     0FDH       ; vuelta al 8085
                          END
```

Figura 7.5: Ejemplo 3: Programa de eco para la E/S.

Comentarios: Este ejemplo muestra una forma de programar el 8088 para que reciba la entrada de un teclado o de un terminal y envíe una salida a un terminal o una impresora. Primero espera, y cuando le llega, obtiene un carácter del dispositivo de entrada; a continuación espera a que el dispositivo de salida esté preparado y le envía este carácter.

El programa utiliza controles de bucle más avanzados. Es decir, el programador es más consciente del uso de estas instrucciones que en el caso de las de control simple de bucles.

Al comienzo del programa hay que preparar las constantes IPSTAT, IPDATA, OPSTAT, OPDATA, IMASK, OMASK,

INITP e INITM de acuerdo con cada computadora. Los saltos condicionales como JNZ deben invertirse en este ejemplo para que pueda ejecutarse en su computadora. Discutiremos estos detalles un poco más adelante. Este segmento de código puede cargarse en la dirección central estándar, la 1000H.

La primera instrucción carga el acumulador (AL, 8 bits) con un modelo de bits que se utilizará para inicializar el controlador de E/S. Su computadora particular puede necesitar un modelo de bits distinto o incluso una serie de modelos (véase el estudio del PSIC 8251 en el capítulo 6). La segunda instrucción coloca este modelo en el port de control del controlador de E/S. También es posible que el lector tenga un port diferente, o tal vez tenga varios ports. Puede ser que no necesite inicialización, o que necesite una rutina más larga. Puede consultar el manual de su sistema de E/S para ver qué es lo que necesita su máquina en particular.

La sección de código siguiente entra un carácter del terminal o teclado. Primero transfiere el contenido del port de estado para la entrada al AL. En la mayoría de las máquinas, uno de los 8 bits del octeto de estado indica si la entrada está disponible. En este caso se trata del bit número 3 y se puede utilizar la instrucción TEST cuya fuente (en el modo de direccionamiento inmediato) es igual al modelo de bits con un 1 en la posición 3 y un 0 en las demás. Este modelo recibe el nombre de máscara debido a que ayuda a *enmascarar* todos los bits salvo el deseado. En este caso la máscara corresponde al número 8. La figura 7.6 muestra cómo funciona.

```
7 6 5 4 3 2 1 0
0 0 0 0 1 0 0 0  =  08H
```

Figura 7.6: Cálculo del valor de la máscara.

Su computadora particular utilizará probablemente otro bit y, por tanto, una máscara distinta.

En nuestra computadora, un 0 en la posición 3 indica que la entrada está disponible, y un 1 que no lo está. Por tanto utilizaremos la instrucción JNZ para volver al bucle cuando el teclado o terminal no está preparado. Si su computadora tiene la forma inversa necesitará la instrucción JZ.

Las instrucciones de TEST y salto condicional trabajan juntas de la siguiente manera: la instrucción TEST hace un producto lógico (AND) bit a bit de la fuente IMASK y el destino AL, y almacena el resultado en una posición temporal dentro de la CPU, en principio desconocida para el programador. Todas las instrucciones aritméticas y lógicas (incluida ésta) actualizan los indicadores según el resultado de la operación. Es decir, si el resultado es cero, ponen a 1 el indicador de cero (ZF), si el resultado es negativo, ponen a 1 el indicador de signo (SF), y así sucesivamente. En este caso nos interesa el indicador de cero. Si el bit de estado es 0 (preparado), el producto lógico del octeto de

estado con la máscara será 0, y el indicador de cero se pondrá a 1. Por lo contrario, si el bit de estado es 1 (no preparado), el producto lógico del octeto de estado con la máscara coincidirá con la máscara, y por tanto el indicador de cero estará a 0. Resumiendo, ZF = 1 indica preparado y ZF = 0 indica no preparado[2].

Los saltos condicionales están diseñados para «engranar» con esta instrucción TEST. Esto es, deberíamos mirar la secuencia TEST/Salto como una unidad. Si el bit especificado era 0, JZ producirá un salto y JNZ *«fracasará»* y se continuará con la siguiente instrucción en secuencia. Por otro lado, si el bit especificado no era 0, JZ será la que no causará salto y JNZ lo producirá; es decir, JZ genera un salto sobre resultado nulo, y JNZ salta cuando el resultado es no-nulo. En cada caso, la combinación de las instrucciones TEST y salto condicional producen bifurcaciones dependiendo del valor del bit 3 del octeto obtenido del port de estado de la entrada.

Esto es, JZ causará un salto si este bit es 0 (preparado) y JNZ causará salto sólo si el bit es 1 (no preparado). JNZ se usa para volver atrás hasta que esté preparado.

La siguiente instrucción toma el carácter que ya está esperando en el port de entrada. El carácter se guarda en AH, debido a que AL se reserva para otros trabajos posteriores.

A continuación, con la instrucción CMP vemos si el carácter es un control C. Si lo es, bifurcaremos a una instrucción que pasa control al 8085.

La salida trabaja de una forma similar a la entrada. En este caso, se pregunta por el mismo port de estado, pero por un bit distinto. El bit 2 es el que contiene el estado de la salida. La máscara de estado de salida es 00000100 en binario, que es 04H. Nótese que colocamos instrucciones EQU al principio del programa para preparar estos valores. Es una buena costumbre hacerlo así porque si queremos modificarlos, están todos juntos y se necesita sólo un cambio para dar los valores correctos en cada ocurrencia de cualquier constante dada a lo largo del resto del código.

De nuevo para la salida, un 1 significa no preparado, y un 0 preparado. Por ello hemos escogido la instrucción JNZ como salto condicional al principio del bucle hasta que el dispositivo de salida esté preparado. Cuando lo está, se mueve el código ASCII del carácter del port de salida. Como antes, el lector deberá dar su valor particular de la dirección de este port.

El método de E/S anterior utiliza lo que se llama el sondeo periódico (se vio junto con el IOP 8089 en el capítulo 5). La computadora se queda en un ciclo de espera preguntando cada vez si el dispositivo está preparado, y no hace ningún trabajo útil durante la espera.

---

[2] Nótese que el indicador ZF es exactamente el opuesto del resultado. Esto, aunque pueda llevar a confusión, es lógicamente correcto. Como regla nemotécnica, ZF=0 significa *no,* el resultado no es 0, y ZF=1 significa *sí,* el resultado es 0.

Llamada y retorno de subrutinas

Para que los programas resulten eficientes y legibles tanto en lenguaje ensamblador como en lenguaje de alto nivel, resultan indispensables las subrutinas. Las instrucciones que entran en este grupo son:

CALL	Llamada
RET	Vuelta

La figura que sigue muestra cómo se pueden implementar algunas subrutinas básicas de E/S en un sistema de computadora. Mostraremos las subrutinas y cómo son llamadas. En este ejemplo no se usa ninguna pantalla.

```
                        ; Ejemplo 4: (CALL y RETURN)
                        ;
                        ; Rutinas de E/S
0000'                           ASEG
0080                    IO      EQU     0080H   ; dirección segmento E/S
0020                    CDIN    EQU     0020H   ; desplazamiento rutina de entrada
0026                    CDOUT   EQU     0026H   ; desplazamiento rutina de salida
0000                    ENTER:                  ; bucle principal
0000    9A 0020                 CALL    CDIN,IO ; obtener un carácter
0003    0080
0005    3C 03                   CMPBI   AL,03   ; preguntar por control c
0007    74 07                   JZ      RETURN  ; si es que sí, volver
0009    9A 0026                 CALL    CDOUT,IO ; imprimir el carácter
000C    0080
000E    EB F0                   JMPS    ENTER   ; salto corto al comienzo del bucle
                        ;
0010    E4 FD           RETURN: INB     0FDH    ; vuelta al 8085
                                END             ; fin del segmento de código
```

Figura 7.7a: Ejemplo 4: Utilización de rutinas de E/S.

```
                Sección E/S

                Este segmento contiene 4 rutinas de E/S por consola, que
                pueden ser llamadas por rutinas de otros segmentos.

                Las rutinas son:

;       CDINIT: Esta rutina inicializa el dispositivo
;               de consola. No necesita parámetros de
;               entrada, y no devuelve nada. Preserva
;               todos los registros salvo el AL.
;
;       CDBRK:  Esta rutina pregunta por la entrada
;               de consola. No necesita parámetros de
;               entrada. Si no se oprime ninguna tecla,
;               vuelve con el indicador Z a cero. En caso
;               contrario, pregunta por el control C o el
;               control S. Si no hay ninguno, vuelve con
;               el código ASCII de la clave tecleada en
;               AL y AH. Si control C, vuelve al 8085.
;               Si control S, espera un carácter que
;               devuelve en AL y AH. Preserva todos los
;               demás registros.
;
;       CDIN:   Esta rutina espera un carácter del
;               dispositivo de entrada de la consola.
;               No necesita parámetros de entrada.
;               Vuelve con el código ASCII de la clave en
;               AL y AH. Preserva todos los demás registros.
```

```
                        ;       CDOUT:      Esta rutina envía un carácter al dispositivo
                        ;                   de salida de la consola. Necesita que el
                        ;                   código ASCII del carácter esté en el
                        ;                   registro AH. Preserva todos los demás
                        ;                   registros.
                        ;
0000'                           ASEG
0086            IPSTAT          EQU         86H         ; port de estado de entrada
0083            IPDATA          EQU         83H         ; port de datos de entrada
0080            OPSTAT          EQU         80H         ; port de estado de salida
0082            OPDATA          EQU         82H         ; port de datos de salida
0008            IMASK           EQU         08H         ; máscara de estado de entrada
0004            OMASK           EQU         04H         ; máscara de estado de salida
0086            INITP           EQU         86H         ; port de inicialización
0014            INITM           EQU         14H         ; máscara de inicialización
                ;
0000            CDINIT:                                 ; inicializar consola
0000    B0 14                   MOVBI       AL,INITM    ; máscara de inicialización
0002    E6 86                   OUTB        INITP       ; port de inicialización
0004    CB                      RETS                    ; vuelta intersegmento
                ;
0005            KEY:                                    ; rutina local para la entrada
0005    E4 86                   INB         IPSTAT      ; obtener el estado
0007    A8 08                   TESTBI      AL,IMASK    ; verificarlo
0009    75 04                   JNZ         KEYRET      ; volver si nada
000B    E4 83                   INB         IPDATA      ; sino, obtener el dato
000D    BA E0                   MOVB        AH,AL       ; devolverlo en AH
000F    C3      KEYRET:         RET                     ; vuelta intrasegmento
                ;
0010            CDBRK:                                  ; rutina de ruptura de consola
0010    E8 FFF2                 CALL        KEY         ; preguntar por la clave
0013    75 0A                   JNZ         CDBRET      ; volver si no hay clave
0015    3C 03                   CMPBI       AL,03H      ; ¿control C?
0017    74 18                   JZ          RETURN      ; si es que sí, volver al 8085
0019    3C 13                   CMPBI       AL,13H      ; ¿control S?
001B    74 03                   JZ          CDIN        ; si es que sí, esperar la clave
001D    3A C0                   CMPB        AL,AL       ; poner Z a 1 para indicar la clave
001F    CB      CDBRET:         RETS                    ; vuelta intersegmento
                ;
0020            CDIN:                                   ; rutina de entrada por consola
0020    E8 FFE2                 CALL        KEY         ; preguntar por la clave
0023    75 FB                   JNZ         CDIN        ; esperar hasta que se introduzca
0025    CB                      RETS                    ; vuelta intersegmento
                ;
0026            CDOUT:                                  ; rutina de entrada por consola
0026    E4 80                   INB         OPSTAT      ; estado de la salida
0028    A8 04                   TESTBI      AL,OMASK    ; verificarlo
002A    75 FA                   JNZ         CDOUT       ; realizar un bucle hasta que esté preparado
002C    8A C4                   MOVB        AL,AH       ; obtener el dato preparado
002E    E6 82                   OUTB        OPDATA      ; sacarlo
0030    CB                      RETS                    ; vuelta intersegmento
                ;
0031    E4 FD   RETURN:         INB         0FDH        ; vuelta al 8085
                                END                     ; FIN DEL SEGMENTO
```

Figura 7.7b: Segmento IO.

Comentarios: El ejemplo muestra cómo implementar dos tipos distintos de llamadas y vueltas de subrutinas (intersegmento e intrasegmento).

Hay dos segmentos distintos en el ejemplo; un segmento que contiene el programa principal (en la posición usual 1000H), y un segmento en el que se encuentran las rutinas básicas de E/S. Hemos llamado EIO.MAC al código fuente del segmento IO, y E4.MAC al código fuente del programa principal. El segmento IO debe cargarse en la dirección 800H. Su número de segmento (80H) se iguala (EQU) al símbolo IO en el programa principal,

de forma que se puede referenciar por un *nombre* mejor que por el número.

Se deben tener ambos segmentos en memoria para que el programa funcione.

Probablemente el lector quiera crear un fichero que incluya el L88 (en 100H) y el segmento IO (en 800H). Esto se puede hacer utilizndo la DDT para cargarlo en EIO.COM (Código máquina para segmento IO) y llevarlo al lugar correcto. Utilice las órdenes:

IL88.COM
R

para cargar en L88 mientras todavía esté en DDT. Haga un control C para volver al CP/M y utilice la orden:

SAVE 15 GO.COM

para salvaguardar la región de 100H a 0FFFH como la orden GO. Posteriormente podrá añadir dos segmentos del sistema a esta misma área.

Si ha creado la orden GO anterior, podrá ejecutar éste y los siguientes ejemplos usando DDT para cargar el ejemplo y luego (desde CP/M principal) tecleando GO.

Los símbolos CDINIT, CDBRK, CDIN y CDOUT aparecen en el segmento IO como rótulos para las rutinas IO, y también aparecen como igualadores (EQU) en el programa principal. En la práctica el fabricante debe suministrar el segmento IO y un fichero con los igualadores. Enlazando o editando este fichero el usuario puede hacer todas las referencias necesarias a estas rutinas, por nombre mejor que por dirección mientras escribe los programas de aplicación.

Veamos ahora el programa principal. Llama primero a CDIN, la rutina de entrada por consola, que espera un carácter del dispositivo de entrada de la consola. Fijémonos que la instrucción CALL es una llamada inter-segmento; esto es, es una llamada de un segmento a otro. Tiene dos operandos: la *dirección del desplazamiento* y *el número de segmento* en el que se encuentra. Ambos operandos se referencian por nombre. En el ensamblador cruzado de Microsoft, el desplazamiento siempre aparece primero que el número de segmento, separadas ambas cantidades por una coma.

CDIN devuelve el carácter en AL y AH. Se comprueba si el carácter es un control C (el botón del pánico), y si lo es se salta a la vuelta estándar de control al 8085. Si no lo es, se llama a CDOUT. Esta llamada es también intersegmento (al segmento IO). Acepta el carácter en AH y lo envía al dispositivo de salida. Una vez que se ha enviado al carácter, el programa entra en un bucle de espera hasta la llegada de otro carácter.

El segmento de E/S comienza con una sección muy larga de documentación en la que se describe cada rutina en detalle. Con esta documentación se cubren los siguientes puntos:

a) función o propósito de la rutina
b) parámetros de entrada
c) parámetros de salida
d) registros preservados o alterados

Es muy importante tener este tipo de documentación en rutinas como éstas que van a ser utilizadas por otras personas.

Además de las cuatro rutinas de E/S para uso externo, el segmento I/O contiene una rutina llamada KEY de uso interno, que no puede llamarse con éxito desde ningún otro segmento. Esto se debe a que la instrucción RET de la rutina es una instrucción de vuelta intrasegmento y no intersegmento. La RETS intersegmento restaura el segmento de código y los registros puntero de instrucción, mientras que la RET intrasegmento sólo restaura el puntero de instrucción. De este modo, una llamada a KEY desde otro segmento no volvería bien el segmento de llamada. El último programa nos permitirá ver realmente lo distintas que son ambas llamadas. Podremos recorrer paso a paso el programa viendo la pila para comprobar cómo la instrucción RET *extrae* una cantidad de la pila (devolviendo el registro IP) y cómo la instrucción RETS extrae dos cantidades (devolviendo los registros CS e IP).

El trabajo que realizan las subrutinas de E/S es muy similar al de los procedimientos de E/S descritos en la sección anterior de este capítulo.

Como siempre, el lector necesitará *configurarlas* de acuerdo con su sistema de computadora. Si su sistema no necesita rutina de inicialización, puede reemplazar la que tenemos nosotros por una serie de cuatro NOP. Aunque todavía no hemos introducido las operaciones NOT ni NOP, admita esta sugerencia «off-the-record».

La selección de JZ o JNZ depende de cómo están preparados los bits de los octetos de estado. Esto es una cosa que puede causar serios problemas en sus programas particulares de aplicación. Si tiene los bits invertidos, una forma de cortar de raíz este problema es introducir la instrucción lógica:

```
NOT     AL
```

justo antes de la instrucción TEST. Normalmente se insertan un par de NOP para reservar sitio en el código máquina por si se necesita reemplazar alguno por una instrucción NOT. (De nuevo «off-the-record».)

Un punto peculiar en la rutina CDBRK es que la instrucción

```
CMPB     AL,AL
```

aparece justo antes de una RET. Con esto nos aseguramos que el indicador Z tome el valor correcto para la vuelta, indicando que el carácter está preparado. Si sus bits de estado de la entrada todavía están cambiados, y quiere modificar los saltos condicio-

nales en vez de usar la instrucción NOT, puede cambiar esta instrucción por:

CMPBI AL,03

Instrucciones aritméticas

El grupo de instrucciones aritméticas incluye una buena cantidad de instrucciones:

Grupo de adición

ADD	Suma
ADC	Suma con acarreo
AAA	Ajuste ASCII para la suma
DAA	Ajuste decimal para la suma

Grupo de substracción

SUB	Resta
SBB	Resta con acarreo negativo
AAS	Ajuste ASCII para la resta
DAS	Ajuste decimal para la resta

Grupo de multiplicación

MUL	Multiplicación
IMUL	Multiplicación entera
AAM	Ajuste ASCII para la multiplicación

Grupo división

DIV	División
IDIV	División entera
AAD	Ajuste ASCII para la división

Conversiones

CBW	Pasar octeto a palabra
CWD	Pasar palabra a doble palabra
NEG	Negación

Veremos sólo un par de instrucciones de este tipo en el ejemplo (véase la figura 7.8). Se puede pasar el resto de la vida desarrollando ejemplos para las instrucciones que no veremos.

```
                        ; Ejemplo 5A: (Instrucciones aritméticas)
                        ;
                        ; Conteo multidígito para pantalla
                        ;
0000'                             ASEG
0E80                    VIDEO     EQU     0E80H
000A                    SIZE      EQU     10
0080                    IO        EQU     80H          ; dirección segmento IO
0010                    CKBRK     EQU     10H          ; rutina de ruptura de consola
                        ;
0000    B8 0E80         ENTER:    MOVI    AX,VIDEO     ; apuntar el segmento de datos
0003    8E D8                     MOV     DS,AX        ; a la RAM de visualización
          .             ;
                                                       ; inicializar el número
0005    BF 004F                   MOVI    DI,79        ; apuntar a la primera posición
0008    B9 000A                   MOVI    CX,SIZE      ; CX contiene el contador
000B    B0 30                     MOVBI   AL,'0'       ; AL contiene a '0'
000D    88 05           ENTER1:   MOVB    [DI],AL      ; colocar el dígito en su lugar
000F    4F                        DEC     DI           ; posición siguiente
0010    E2 FB                     LOOP    ENTER1       ; realizar bucle hasta que se haya acabado
                        ;
0012    BF 004F         LOOP:     MOVI    DI,79        ; bucle principal
0015    B9 000A                   MOVI    CX,SIZE      ; inicializar el contador del bucle
0018    8A 25                     MOVB    AH,[DI]      ; usar AH para el acarreo
001A    FE C4                     INCB    AH
001C    8A C4           DLOOP:    MOVB    AL,AH        ; obtener el dígito
```

```
001E  8A 65 FF        MOVB    AH,-1[DI]      ; obtener el dígito siguiente
0021  37              AAA                    ; obtener el nuevo acarreo
0022  04 30           ADDBI   AL,'0'         ; pasarlo a ASCII
0024  88 05           MOVB    [DI],AL        ; devolverlo
0026  4F              DEC     DI             ; posición siguiente
0027  E2 F3           LOOP    DLOOP          ; dígito siguiente
                ;
0029  9A 0010         CALL    CKBRK,IO       ; verificar interrupción
002C  0080
                ;
002E  E2 E2           LOOP    LOOP           ; siempre vuelve al comienzo del bucle
                ;
                      END                    ; fin del segmento código
```

Figura 7.8a: Ejemplo 5A: Conteo multidígito para pantalla.

```
                ; Ejemplo 5B: (Instrucciones aritméticas)
                ;
                ; Conteo multidigito para terminal
                ;
0000'                 ASEG
0E80'           VIDEO EQU     0E80H
0080            IO    EQU     80H            ; dirección segmento IO
0026            COUT  EQU     26H            ; salida de consola
0010            CKBRK EQU     10H            ; ruptura de consola
000A            SIZE  EQU     10
                ;
0000  B8 0E80         MOVI    AX,VIDEO       ; el segmento de datos apunta
0003  8E D8           MOV     DS,AX          ; a la RAM de visualización
                ;
                                             ; inicializar el número
0005  BF 004F         MOVI    DI,79          ; apuntar a la primera posición
0008  B9 000A         MOVI    CX,SIZE        ; CX contiene el contador
000B  B0 30           MOVBI   AL,'0'         ; AL contiene '0'
000D  88 05     ENTER1: MOVB  [DI],AL        ; colocar el dígito en su lugar
000F  4F              DEC     DI             ; posición siguiente
0010  E2 FB           LOOP    ENTER1         ; realizar bucle hasta que se haya acabado
                ;
0012  BF 004F   LOOP: MOVI    DI,79          ; bucle principal
0015  B9 000A         MOVI    CX,SIZE        ; dar valor al contador del bucle
0018  8A 25           MOVB    AH,[DI]        ; usar AH para el acarreo
001A  FE C4           INCB    AH
001C  8A C4     LOOP1: MOVB   AL,AH          ; obtener dígito
001E  8A 65 FF        MOVB    AH,-1[DI]      ; obtener dígito siguiente
0021  37              AAA                    ; ajustar
0022  04 30           ADDBI   AL,'0'         ; volver a ASCII
0024  88 05           MOVB    [DI],AL        ; devolverlo
0026  4F              DEC     DI             ; posición siguiente
0027  E2 F3           LOOP    LOOP1          ; dígito siguiente
                ;
0029  9A 0010         CALL    CKBRK,IO       ; verificar si interrupción
002C  0080
                ;
                                             ; sacar el número
002E  BF 0045         MOVI    DI,79-SIZE     ; apuntar a la posición más alta
0031  B9 000B         MOVI    CX,SIZE+1      ; CX contiene el contador
0034  8A 25     OUT:  MOVB    AH,[DI]        ; obtener dígito en el lugar
0036  47              INC     DI             ; posición siguiente
0037  9A 0026         CALL    COUT,IO        ; enviarlo de salida
003A  0080
003C  E2 F6           LOOP    OUT            ; realizar bucle hasta que se haya hecho
                ;
003E  E2 D2           LOOP    LOOP           ; siempre realizar bucle
                ;
                      END                    ; fin del segmento de código
```

Figura 7.8b: Ejemplo 5B: Conteo multidígito para un terminal.

Comentarios: Hay dos ejemplos, uno para una pantalla y otro para un sistema con sólo un terminal. En cada caso el ejemplo muestra una posible forma de realizar operaciones aritméticas

multi-dígito con el 8088. En este ejemplo en particular se utiliza un contador multi-dígito que se almacena en el mismo sitio en que se visualiza sobre la pantalla (o en el caso 5B, se almacena para carga externa al terminal). Los dígitos se almacenan en código ASCII, un dígito por octeto.

El código está en un segmento que se puede ubicar en la posición de memoria estándar 1000H. El segmento I/O también debe estar presente para que funcione la llamada CKBRK (¿tiene los indicadores correctamente preparados para la E/S?).

El programa comienza inicializando el número cargando 0 en todos los dígitos mediante un pequeño bucle. A continuación viene el bucle principal del programa que se ejecuta cada vez que se decrementa el contador. Este bucle inicializa y ejecuta un bucle menor que se encarga de cada dígito, y al que daremos el nombre de bucle de dígito.

El bucle de dígito utiliza una especie de estructura tubular (pipeline) para procesar los dígitos. Carga el dígito siguiente en el registro AH después de que el dígito en curso se haya colocado en AL. La instrucción AAA (ajuste ASCSS para suma) ajusta el dígito en curso en AL sumándole 6 al cuarteto inferior si es mayor que 9. Cualquier acarreo de salida de este cuarteto se suma automáticamente a AH, que contiene el siguiente dígito (según se ha diseñado el programa). La instrucción AAA también pone a cero el cuarteto superior de AL, de forma que se necesita la instrucción siguiente para volver a sumar el cuarteto superior del código ASCII del dígito. A continuación se coloca el dígito en su lugar y se hace que el registro índice DI apunte al siguiente dígito antes de volver al comienzo del bucle.

Es interesante ver la distribución de tiempos del bucle de dígito. Las unidades de tiempo son ciclos de reloj, de 200 nanosegundos para una CPU de la frecuencia de 5 MHz. La figura 7.9 los resume:

```
                               instrucción      direccionamiento      total
LOOP1:   MOVB    AL,AH         2                Ø                     2
         MOVB    AH,-1[DI]     8                9                     17
         AAA                   4                Ø                     4
         ADDBI   AL,'Ø'        4                Ø                     4
         MOVB    [DI],AL       9                5                     14
         DEC     DI            2                Ø                     2
         LOOP    LOOP1         9                Ø                     9
```

Figura 7.9: Ciclos de reloj para el bucle de dígito.

En total se necesitan 52 ciclos de reloj, o lo que es igual, 10,4 microsegundos. Si hay 10 dígitos, el proceso ocupará del orden de 100 microsegundos cada conteo, lo que significa unos 10.000 conteos por segundo. Realmente nosotros medimos 3.700 conteos por segundo. Hay al menos tres razones que justifiquen esta pérdida de eficacia. Una es que la cola de instrucciones no siempre puede obtener instrucciones a la misma velocidad a la que se ejecutan; otra es que la RAM de video puede disminuir la

velocidad de la CPU llevándola a estados de espera; y la tercera es la cantidad de trabajo extra debido al bucle mayor.

Utilizando operaciones de tratamiento de cadenas que veremos a continuación puede aumentarse la velocidad en un 20 por 100.

Las instrucciones de tratamiento de cadenas permiten el movimiento, comparación o búsqueda rápida en bloques de datos. Incluye este grupo las siguientes instrucciones:

Tratamiento de cadenas

MOVC	Transferir carácter de una cadena
MOVW	Transferir palabra de una cadena
CMPC	Comparar carácter de una cadena
CMPW	Comparar palabra de una cadena
SCAC	Buscar carácter de una cadena
SCAW	Buscar palabra de una cadena
LODC	Cargar carácter en una cadena
LODW	Cargar palabra de una cadena
STOC	Guardar carácter de una cadena
STOW	Guardar palabra de una cadena
REP	Repetir
CLD	Poner a 0 el indicador de dirección
STD	Poner a 1 el indicador de dirección

Vimos estas instrucciones extensamente en el capítulo 3, de manera que omitimos una explicación general aquí. En caso de necesitarlo, repase el capítulo 3. La figura 7.10 muestra nuestro programa-ejemplo para este grupo de instrucciones.

```
                         ; Ejemplo 6: (Operadores de tratamiento de cadenas)
                         ;
                         ; Sacar mensajes
                         ;
0000'                              ASEG
0E80                     VIDEO     EQU      0E80H              ; segmento RAM de visualización
0080                     IO        EQU      80H                ; segmento IO
0020                     CDIN      EQU      20H                ; rutina de entrada por consola
0026                     CDOUT     EQU      26H
007F                     DEL       EQU      7FH                ; código ASCII para la tecla DELETE
0020                     SPACE     EQU      20H                ; código ASCII para la tecla SPACE
000A                     LF        EQU      0AH                ; código ASCII para la tecla LINEFEED
000D                     CR        EQU      0DH                ; código ASCII para la tecla ETURN
                         ;
0000                     ENTER:                                ; entrar aquí
0000      BC C8                    MOV      AX,CS              ; segmento de datos igual
0002      8E D8                    MOV      DS,AX              ; al segmento código
                         ;
0004      BE 0058        LOOP:     MOVI     DI,MESS0           ; bucle principal
0007      E8 003C                  CALL     MESOUT             ; mostrar la tabla de opciones
                         ;
000A      9A 0020        LOOP1:    CALL     CDIN,IO            ; obtener la opción
000D      0080
000F      9A 0026                  CALL     CDOUT,IO           ; y mostrarla
0012      0080
```

```
0014     8A C4        ;            MOVB     AL,AH
0016     3C 03                     CMPBI    AL,3               ; ¿control C?
0018     74 3C                     JZ       RETURN             ; en caso afirmativo, volver
                      ;
001A     3C 31                     CMPBI    AL,'1'             ; ¿número 1?
001C     74 10                     JZ       DO1                ; enviar dicho mensaje
                      ;
001E     3C 32                     CAMPBI   AL,'2'             ; ¿número 2?
0020     74 14                     JZ       DO2                ; enviar dicho mensaje
                      ;
0022     3C 33                     CMPBI    AL,'3'             ; ¿número 3?
0024     74 18                     JZ       DO3                ; enviar dicho mensaje
0026     BE 0160      ERROR:       MOVI     SI,MESS4           ; ninguno de los anteriores
0029     E8 001A                   CALL     MESOUT
002C     EB DG                     JMPS     LOOP               ; volver al comienzo del bucle a por más
                      ;
002E     BE 00A9      DO1:         MOVI     SI,MESS1           ; tratar n.º 1
0031     E8 0012                   CALL     MESOUT
0034     EB D4                     JMPS     LOOP1
0036     BE 0101      DO2:         MOVI     SI,MESS2           ; tratar n.º 2
0039     E8 000A                   CALL     MESOUT
003C     EB CC                     JMPS     LOOP1
003E     BE 0122      DO3:         MOVI     SI,MESS3           ; tratar n.º 3
0041     E8 0002                   CALL     MESOUT
0044     EB C4                     JMPS     LOOP1
                      ;
0046                  MESOUT:                                  ; rutina de salida de mensajes
0046     FC                        CLD                         ; dirección hacia adelante
0047     AC           MESLP:       LODC                        ; obtener el carácter
0048     A8 FF                     TESTBI   AL,0FFH            ; un octeto cero significa
004A     74 09                     JZ       MESEND             ; fin de mensaje
004C     8A E0                     MOVB     AH,AL              ; enviar el carácter
004E     9A 0026                   CALL     CDOUT,IO
0051     0080
0053     EB F2                     JMPS     MESLP              ; realizar bucle a por más
0055     C3           MESEND:      RET                         ; volver al final
                      ;
0056     E4 FD        RETURN:      INB      0FDH
                      ;
0058     0D 0A 54 79  MESS0:       DB       CR,LF,'Type one of the following numbers:'
005C     70 65 20 6F
0060     6E 65 20 6F
0064     66 20 74 68
0068     65 20 66 6F
006C     6C 6C 6F 77
0070     69 6E 67 20
0074     6E 75 6D 62
0078     65 72 73 3A
007C     0D 0A 20 20               DB       CR,LF,' 1, 2, or 3'
0080     31 2C 20 32
0084     2C 20 6F 72
0088     20 33
008A     0D 0A 6F 72               DB       CR,LF,' or type control C to exit.'
008E     20 74 79 70
0092     65 20 63 6F
0096     6E 74 72 6F
009A     6C 20 43 20
009E     74 6F 20 65
00A2     78 69 74 2E
00A6     0D 0A 00                  DB       CR,LF,0
                      ;
00A9     0D 0A 23 20  MESS1:       DB       CR,LF,'# 1, Congratulations,'
00AD     31 2E 20 43
00B1     6F 6E 67 72
00B5     61 74 75 61
00B9     74 69 6F 6E
00BD     73 2C
00BF     20 79 6F 75               DB       ' you have made an excellent choice.',CR,LF
00C3     20 68 61 76
00C7     65 20 6D 61
00CB     64 65 20 61
00CF     6E 20 65 78
00D3     63 65 6C 6C
```

```
00D7   65 6E 74 20
00DB   63 68 6F 69
00DF   63 65 2E 0D
00E3   0A
00E4   4E GF 77 20              DB        'Now try the other choices.',CR,LF,0
00EB   74 72 79 20
00EC   74 68 65 20
00F0   6F 74 68 65
00F4   72 20 63 68
00F8   6F 69 63 65
00FC   73 2E 0D 0A
0100   00
                       ;
0101   0D 0A 23 20     MESS2:   DB        CR,LF,'# 2. Poor choice, try asain.',CR,LF,0
0105   32 2E 20 50
0109   6F 6F 72 20
010D   63 68 6F 69
0111   63 65 2C 20
0115   74 72 79 20
0119   61 67 61 69
011D   6E 2E 0D 0A
0121   00
                       ;
0122   0D 0A 23 20     MESS3:   DB        CR,LF,'# 3. Number 1 would be better,',CR,LF
0126   33 2E 20 4E
012A   75 6D 62 65
012E   72 20 31 20
0132   77 6F 75 6C
0136   64 20 62 65
013A   20 62 65 74
013E   74 65 72 2E
0142   0D 0A
0144   57 68 79 20              DB        'Why don''t you try it now?,CR,LF,0
0148   64 6F 6E 27
014C   74 20 79 6F
0150   75 20 74 72
0154   79 20 69 74
0158   20 6E 6F 77
015C   3F 0D 0A 00
                       ;
0160   0D 0A 59 6F     MESS4:   DB        CR,LF,'You didn''t hit the right key!!!'
0164   75 20 64 69
0168   64 6E 27 74
016C   20 68 69 74
0170   20 74 68 65
0174   20 72 69 67
0178   68 74 20 6B
017C   65 79 21 21
0180   21
0181   0D 0A 00                 DB        CR,LF,0
                       ;
                                END
```

Figura 7.10: Ejemplo 6: Sacar mensajes.

Comentarios: Este programa está diseñado un poco como entretenimiento. El programa visualiza un mensaje que le pide un número del 1 al 3. Cada número hace que aparezca un mensaje distinto sobre la pantalla. Se tiene también la opción de hacer un control C para volver al cargador. Los mensajes se limitan a hacerle dar vueltas y más vueltas, pero se puede modificar fácilmente para que los mensajes de salida sean útiles. Este programa podría ser el comienzo de un juego, o de otro programa que realizará funciones útiles. El programa utiliza la instrucción de tratamiento de cadenas LODC.

Este programa puede ubicarse en la dirección central estándar

1000H, y necesita que el segmento I/O esté ubicado en su dirección estándar 800H. Si tiene creado la orden GO, se hace automáticamente.

Hay un bucle principal que gestiona el programa y llama a la subrutina MESOUT para sacar mensajes, y a la subrutina CDOUT para entrar pulsaciones del usuario. MESOUT está en el mismo segmento y CDOUT en otro (en el I/O). La instrucción CMP y los saltos condicionales se usan para enviar las distintas opciones.

En la rutina MESOUT se utiliza el registro SI para apuntar al carácter que se va a sacar. La instrucción LODC transfiere el carácter de memoria al acumulador e incrementa SI para que apunte al siguiente carácter. Antes de sacar el carácter se mira si es cero. En nuestro programa, el cero indica el fin del mensaje. Nótese que cada mensaje tiene un cero pegado a él. Algunos ensambladores tienen en realidad una orden especial que coloca automáticamente un cero al final del mensaje. Hay dos formas más estándar de terminar los mensajes, pero hemos preferido ésta. Fijémonos que los retornos de carro y saltos de líneas son parte del mensaje.

Todos los mensajes se almacenan al final del programa, con una estructura elegante. Podríamos, desde luego, reservar un segmento completo para los mensajes. En cualquier caso el registro de segmentación de datos (DS) deberá apuntar al segmento que contenga los mensajes. En este programa concreto esto se consigue transfiriendo el contenido del registro de segmentación de código (CS) al registro de segmentación de datos.

Instrucciones lógicas

Las instrucciones lógicas son operaciones bit a bit que trabajan sobre octetos o palabras completas. Incluye este grupo las siguientes instrucciones:

NOT	NOT (negación)
AND	AND (producto lógico)
OR	OR (suma lógica)
XOR	OR exclusiva

La figura 7.11 presenta un programa-ejemplo en el que se usan las operaciones lógicas para controlar un cursor mientras se están tecleando caracteres en la pantalla. Si no posee pantalla de visualización no podrá comprobar el funcionamiento de este ejemplo. En tal caso, estudie el programa y pase al siguiente, que también utiliza instrucciones lógicas.

```
                        ; Ejemplo 7: (Operadores lógicos)
                        ;
                        ; Visualizar una línea por pantalla
                        ;
0000'                           ASEG
0E80            VIDEO           EQU     0E80H   ; segmento RAM de video
00B0            PARM            EQU     0B0H    ; segmento de parámetros de los datos
0000            LINELOC         EQU     0       ; dirección del comienzo
                                                ; de línea
```

```
0080                IO       EQU    80H             ; segmento IO
0020                CDIN     EQU    20H             ; rutina de entrada por consola
007F                DEL      EQU    7FH             ; código ASCII para la tecla DELETE
0020                SPACE    EQU    20H             ; código ASCII para la tecla SPACE
000D                CR       EQU    0DH             ; código ASCII para la tecla RETURN

0000                ENTER:                          ; bucle principal
0000   B8 0E80               MOVI   AX,VIDEO        ; el segmento extra es la RAM de video
0003   8E C0                 MOV    ES,AX
0005   B8 00B0               MOVI   AX,PARM         ; el segmento de datos apunta a los datos
0008   8E D8                 MOV    DS,AX
000A   C7 06 0000            MOVI   LINELOC,0       ; inicializar LINELOC
000E   0000

0010   8B 3E 0000   LINEBG;  MOV    DI,LINELOC      ; DI contiene la dirección de comienzo
0014   FC                    CLD                    ; dirección hacia adelante

0015   26           LOOP:    $ES                    ; ES apunta a la RAM de video
0016   80 0D 80              ORBI   [DI],80H        ; poner el cursor
0019   9A 0020               CALL   CDIN,IO         ; obtener un carácter
001C   0080
001E   26                    $ES                    ; ES apunta a la RAM de video
001F   80 25 7F              ANDBI  [DI],7FH        ; eliminar cursor
0022   3C 03                 CMPBI  AL,3            ; preguntar por el control C
0024   74 1F                 JZ     RETURN          ; en caso afirmativo, ir al 8085
0026   3C 7F                 CMPBI  AL,DEL          ; preguntar por DELETE
0028   74 07                 JZ     DELETE
002A   3C 0D                 CMPBI  AL,CR           ; preguntar  por <CR>
002C   74 10                 JZ     NXTLN           ; en caso afirmativo, línea nueva
                                                    ; si ninguno de los anteriores,
002E   AA                    STOC                   ; colocar el carácter en su lugar
                                                    ;
002F   EB E4                 JMPS   LOOP            ; siempre se vuelve al comienzo del bucle

0031                DELETE:                         ; procedimiento para gestionar DELETE
0031   3B 3E 0000            CMP    DI,LINELOC      ; preguntar por el comienzo de línea
0035   74 DE                 JZ     LOOP
0037   4F                    DEC    DI              ; copia de seguridad
0038   26                    $ES
0039   C6 05 20              MOVBI  [LI],SPACE      ; ponerlo a blancos
003C   EB D7                 JMPS   LOOP            ; y volver al comienzo del bucle
003E                NXTLN:                          ; procedimiento para obtener nueva línea
003E   83 06 0000 50         ADDI   LINELOC,80      ; nuevo comienzo de línea
0043   EB CB                 JMPS   LINEBG          ; y vuelta al bucle

0045   E4 FD        RETURN:  INB    OFDH            ; vuelta al 8085
                             END                    ; FIN DEL SEGMENTO CODIGO
```

Figura 7.11: Ejemplo 7: Visualizar una línea por pantalla.

Comentarios: El ejemplo muestra cómo convertir la pantalla de visualización en un terminal. Las operaciones de tratamiento de cadenas y las operaciones lógicas AND y OR juegan un papel importante en la visualización de letras y del cursor.

Este segmento se ubica en la dirección estándar 1000H. El segmento de E/S también debe estar en su posición habitual, la 800H.

Hay un bucle principal con un par de procedimientos de servicio. El segmento extra apunta a la pantalla, y el segmento de datos al área de almacenamiento de variables.

La primera línea de texto comienza en la posición 0 del segmento VIDEO. El registro índice DI contiene la posición del cursor. El bucle coloca un tipo de cursor de video inverso sobre la pantalla realizando una OR lógica entre el modelo de bits 10000000 y la posición en curso del cursor. Con esta operación se pone a 1 el bit más significativo del carácter visualizado. A continuación se espera a que llegue otro carácter desde el teclado.

Tan pronto como se recibe este carácter, se desactiva el cursor haciendo la AND lógica con modelo de bits 01111111. Se mira si el carácter es un DELETE, ⟨CR⟩, o control C. Si no es ninguno de éstos, se coloca el carácter sobre la pantalla en la posición que ocupaba el cursor mediante la instrucción STOC. Esto incrementa automáticamente el valor de DI. Las instrucciones AND y OR necesitan instrucciones de anulación de segmento, de forma que para acceder a los destinos se prefiere utilizar el segmento extra al segmento de datos.

El procedimiento de servicio para gestionar el DELETE mira si se está al comienzo de una línea. Si se está, se impide que el usuario vaya más atrás. Si no se está al principio de línea, se puede borrar, cosa que se hace decrementando DI, llevando hacia atrás el cursor, y transfiriendo un espacio a dicha posición. De nuevo, el prefijo de anulación de segmento se usa para apuntar al segmento extra.

El procedimiento de servicio que gestiona ⟨CR⟩ suma 80 a la variable que contiene la dirección de memoria del comienzo de la línea, yendo de esta manera a una nueva línea. Entra entonces en un bucle justo antes del bucle normal en el que se inicializa el registro índice DI para que apunte al comienzo de la (nueva) línea. Si se tiene un número de caracteres por línea distinto, habrá que sumar dicho número en vez de 80.

El procedimiento de servicio que gestiona el control C es sencillamente una vuelta normal a la CPU 8085.

A partir de este esquema básico, animamos al lector a que intente aumentar el número de funciones especiales. Por ejemplo, podría añadir la posibilidad de «scrolling» (posibilidad de que el texto visualizado suba en la pantalla conforme se van escribiendo nuevas líneas), utilizando la instrucción MOVC para subir la pantalla una línea.

Instrucciones de desplazamiento, rotaciones y adeudos

Las instrucciones de desplazamiento, rotación, y adeudos pueden ser útiles en multitud de casos dentro del lenguaje ensamblador. Básicamente, permiten multiplicar y dividir por potencias de 2. En este grupo se incluyen las siguientes instrucciones.

SHL SAL	Desplazar a la izquierda (desplazamiento aritmético)
SHR	Desplazar a la derecha
SAR	Desplazamiento aritmético a la derecha
ROL	Rotación a la izquierda
ROR	Rotación a la derecha
RCL	Rotación con acarreo a la izquierda
RCR	Rotación con acarreo a la derecha
CLC	Borrar acarreo
STC	Poner acarreo a 1

La figura siguiente muestra un ejemplo de cómo utilizar estas instrucciones para preparar los números para la salida.

```
                         ; Ejemplo 8: (Instrucciones de desplazamientos y rotaciones)
                         ;
                         ; Aritmética hexadecimal
                         ;
0000'                             ASEG
0020                     SPACE    EQU     20H              ; código ASCII para SPACE
000D                     CR       EQU     0DH              ; código ASCII para RETURN
000A                     LF       EQU     0AH              ; código ASCII para LINEFEED
0080                     IO       EQU     80H              ; segmento IO
0010                     CKBRK    EQU     10H              ; desplazamiento de CKBRK
0026                     CDOUT    EQU     26H              ; desplazamiento de CDOUT
0090                     SYS1     EQU     90H              ; segmento SYS1
0000                     HEXOUT   EQU     00H              ; desplazamiento de HEXOUT
001B                     HEXIN    EQU     1BH              ; desplazamiento de HEXIN
004B                     SPCOUT   EQU     4BH              ; desplazamiento de SPCOUT
0053                     CRLF     EQU     53H              ; desplazamiento de CRLF
0062                     MESOUT   EQU     62H              ; desplazamiento de MESOUT
                         ;
0000    8C C8            ENTER:   MOV     AX,CS            ; segmento de datos igual
0002    8E D8                     MOV     DS,AX            ; al segmento código
                         ;
0004    BE 0090                   MOVI    SI,MESS0         ; mensaje de bienvenida
0007    9A 0062                   CALL    MESOUT,SYS1
000A    0090
                         ;
000C    9A 0053          LOOP:    CALL    CRLF,SYS1
000F    0090
0011    BE 00FA                   MOVI    SI,MESS1         ; ¿primer número?
0014    9A 0062                   CALL    MESOUT,SYS1
0017    0090
0019    9A 001B                   CALL    HEXIN,SYS1       ; introducir los números
001C    0090
001E    89 16 008C                MOV     NUM1,DX
                         ;
0022    9A 0053                   CALL    CRLF,SYS1
0025    0090
0027    BE 010A                   MOVI    SI,MESS2         ; ¿segundo número?
002A    9A 0062                   CALL    MESOUT,SYS1
002D    0090
002F    9A 001B                   CALL    HEXIN,SYS1       ; introducir los números
0032    0090
0034    89 16 008E                MOV     NUM2,DX
                         ;
0038    9A 0053                   CALL    CRLF,SYS1
003B    0090
003D    BE 011A                   MOVI    SI,MESS3         ; la suma
0040    9A 0062                   CALL    MESOUT,SYS1
0043    0090
0045    8B 16 008C                MOV     CX,NUM1
0049    03 16 008E                ADD     DX,NUM2
004D    9A 0000                   CALL    HEXOUT,SYS1
0050    0090
                         ;
0052    BE 0124                   MOVI    SI,MESS4         ; la diferencia
0055    9A 0062                   CALL    MESOUT,SYS1
0058    0090
005A    8B 16 008C                MOV     DX,NUM1
005E    2B 16 008E                SUB     DX,NUM2
0062    9A 0000                   CALL    HEXOUT,SYS1
0065    0090
                         ;
0067    BE 0137                   MOVI    SI,MESS5         ; el producto
006A    9A 0062                   CALL    MESOUT,SYS1
006D    0090
006F    A1 008C                   MOV     AX,NUM1
0072    F7 26 008E                MUL     NUM2
0076    8B D0                     MOV     DX,AX
0078    9A 0000                   CALL    HEXOUT,SYS1
007B    0090
                         ;
007D    9A 0053                   CALL    CRLF,SYS1
0080    0090
0082    BE 0147                   MOVI    SI,MESSG         ; algunas estrellas
0085    9A 0062                   CALL    MESOUT,SYS1
0088    0090
                         ;
008A    EB 80                     JMPS    LOOP             ; volver al comienzo del bucle
                         ;
008C    0000             NUM1:    DW      0                ; almacenamiento para el número
008E    0000             NUM2:    DW      0                ; almacenamiento para el número
```

```
0090    54 68 69 73     MESS0:   DB      'This program does hexadecimal arithmetic.'
0094    20 70 72 6F
0098    67 72 61 6D
009C    20 64 6F 65
00A0    73 20 68 65
00A4    78 61 64 65
00AB    63 69 6D 61
00AC    6C 20 61 72
00B0    69 74 68 6D
00B4    65 74 69 63
00B8    2E
00B9    0D 0A 49 74              DB      CR,LF,'It computes the sum, difference,
00BD    20 63 6F 6D
00C1    70 75 74 65
00C5    73 20 74 68
00C9    65 20 73 75
00CD    6D 2C 20 64
00D1    69 66 66 65
00D5    72 65 6E 63
00D9    65 2C 20
00DC    61 6E 64 20              DB      'and product of two numbers.',CR,LF,0
00E0    70 72 6F 64
00E4    75 63 74 20
00E8    6F 66 20 74
00EC    77 6F 20 6E
00F0    75 6D 62 65
00F4    72 73 2E 0D
00F8    0A 00
00FA    46 69 72 73     MESS1:   DB      'First number: ',0
00FE    74 20 6E 75
0102    6D 62 65 72
0106    3A 20 20 00
010A    53 65 63 6F     MESS2:   DB      'Second number: ',0
010E    6E 64 20 6E
0112    75 6D 62 65
0116    72 3A 20 00
011A    54 68 65 20     MESS3:   DB      'The sum: ',0
011E    73 75 6D 3A
0122    20 00
0124    20 20 54 68     MESS4:   DB      'The difference: ',0
0128    65 20 64 69
012C    66 66 65 72
0130    65 6E 63 65
0134    3A 20 00
0137    20 20 54 68     MESS5:   DB      ' The product: ',0
013B    65 20 70 72
013F    6F 64 75 63
0143    74 3A 20 00
0147    2A 2A 2A 2A     MESS6: DB DB     '************************************',0
014B    2A 2A 2A 2A
014F    2A 2A 2A 2A
0153    2A 2A 2A 2A
0157    2A 2A 2A 2A
015B    2A 2A 2A 2A
015F    2A 2A 2A 2A
0163    2A 2A 2A 2A
0167    2A 2A 2A 2A

                ;
                         END                      ; Fin del segmento de código
```

Figura 7.12a: Ejemplo 8: Aritmética hexadecimal.

```
; Segmento del sistema #1
;
; Este segmento contiene rutinas de salida de números
; hexadecimales, espacios y vueltas de carro - saltos de línea.
; Este segmento necesita el segmento de E/S.
;
; Las rutinas son:
;
;       HEXOUT:   Saca el contenido del registro DX
;                 como un número hexadecimal de 16 bits
;                 por consola. Necesita un parámetro de
;                 entrada en el registro DX. No devuelve nada.
;                 Destruye los registros AX, BX, CX y DX.
;
```

```
                  ;        HEXIN:    Entra un número hexadecimal de 16 bits
                  ;                  de la consola. Visualiza los dígitos
                  ;                  conforme se van escribiendo.
                  ;                  Devuelve el número en el registro DX.
                  ;                  Destruye los registros AX, BX, CX y DX.
                  ;                  Si se hace un control C, vuelve al
                  ;                  programa cargador.
                  ;
                  ;        SPCOUT:   Saca un espacio por consola. No necesita
                  ;                  parámetros de entrada, y no devuelve nada.
                  ;                  Destruye el registro AX.
                  ;
                  ;        CRLF:     Saca un <CR> <LF> por consola. No necesita
                  ;                  parámetros de entrada, y no devuelve nada.
                  ;                  Destruye el registro AX.
                  ;
                  ;        MESOUT:   Saca un mensaje por consola. El registro SI
                  ;                  debe apuntar al comienzo del mensaje.
                  ;                  El fin de mensaje se reconoce por
                  ;                  un octeto cero.
                  ;
0000'                         ASEG
0080              IO          EQU     0080H             ; segmento de E/S
0010              CKBRK       EQU     0010H             ; desplazamiento de CKBRK
0020              CDIN        EQU     0020H             ; desplazamiento de CDIN
0026              CDOUT       EQU     0026H             ; desplazamiento de CDOUT
0020              SPACE       EQU     20H               ; ASCII del espacio
000D              CR          EQU     0DH               ; ASCII de <CR>
000A              LF          EQU     0AH               ; ASCII de <LF> #
                  ;
0000              HEXOUT:                               ; rutina de salida hexadecimal
0000    BB 0004               MOVI    BX,4              ; n.º de dígitos hexadecimales
0003    B1 04     HLOOP:      MOVBI   CL,4              ; un contador de 4 para cada uno
0005    D3 C2                 ROL     DX,CL             ; para desplazar a la izquierda
0007    8B C2                 MOV     AX,DX             ; lo lleva a AX
0009    24 0F                 ANDBI   AL,0FH            ; enmascararlo
000B    27                    DAA                       ; suma 6 si A-F
000C    04 F0                 ADDBI   AL,0F0H           ; sacar adeudo si A-F
000E    14 40                 ADCBI   AL,040H           ; aquí está el ASCII
0010    8A E0     HLOOP1:     MOVB    AH,AL
0012    9A 0026               CALL    CDOUT,IO          ; imprimir el dígito
0015    0080
0017    4B                    DEC     BX
0018    75 E9                 JNZ     HLOOP             ; dígito siguiente
001A    CB                    RETS                      ; fin de HEXOUT
                  ;
001B              HEXIN:                                ; rutina de entrada hexadecimal
001B    BA 0000               MOVI    DX,0              ; inicializar el número
001E    9A 0020   HESIN1:     CALL    CDIN,IO           ; entrar un dígito
0021    0080
0023    3C 03                 CMPBI   AL,3              ; ¿control C?
0025    74 4B                 JZ      RETURN
0027    9A 0026               CALL    CDOUT,IO          ; en caso contrario, hacer el eco
002A    0080
                  ;
002C    8A C4                 MOVB    AL,AH             ; y
002E    2C 30                 SUBBI   AL,'0'            ; pasar de ASCII
0030    72 18                 JB      HEXRET            ; ¿demasiado bajo?
0032    3C 0A                 CMPBI   AL,'9'-'0'+1      ; ¿de 0 a 9?
0034    72 0A                 JB      HEXADD            ; en caso afirmativo, tomarlo
0036    2C 11                 SUBBI   AL,'A'-'0'        ; ¿de A a F?
0038    72 10                 JB      HEXRET            ; en caso negativo, volver
003A    3C 06                 CMPBI   AL,'F'-'A'+1
003C    73 0C                 JNB     HEXRET
003E    04 0A                 ADDBI   AL,10             ; en caso afirmativo, ajustar
                  ;
0040    B1 04     HEXADD:     MOVBI   CL,4              ; ahora
0042    D3 E2                 SAL     DX,CL             ; multiplicar la parte antigua por 16
0044    B4 00                 MOVBI   AH,0              ; y sumarle la parte nueva
0046    03 D0                 ADD     DX,AX
0048    EB D4                 JMPS    HEXIN1
                  ;
004A    CB        HEXRET:     RETS                      ; volver cuando se acabe
                  ;
004B              SPCOUT:                               ; rutina de salida de espacios
004B    B4 20                 MOVBI   AH,SPACE          ; espacio
004D    SA 0026               CALL    CDOUT,IO          ; sacarlo
0050    0080
0052    CB                    RETS                      ; fin de SPOUT
                  ;
```

```
0053                    CRLF:
0053      B4 0D                   MOVBI     AH,CR          ; vuelta de carro
0055      9A 0026                 CALL      CDOUT,IO       ; sacarlo
0058      0080
005A      B4 0A                   MOVBI     AH,LF          ; nueva línea
005C      9A 0026                 CALL      CDOUT,IO       ; sacarlo
005F      0080
0061      CB                      RETS                     ; fin de CRLF
                        ;
0062                    MESOUT:                            ; rutina de salida de mensajes
0062      FC                      CLD                      ; dirección hacia adelante
0063      AC            MESLP:    LODC                     ; obtener un carácter
0064      A8 FF                   TESTBI    AL,0FFH        ; octeto cero
0066      74 09                   JZ        MESEND         ; fin del mensaje
0068      8A E0                   MOVB      AH,AL          ; enviar el carácter
006A      9A 0026                 CALL      CDOUT,IO
006D      0080
006F      EB F2                   JMPS      MESLP          ; volver al comienzo del bucle a por más
                        ;
0071      CB            MESEND:   RETS                     ; volver al final
                        ;
0072      E4 FD         RETURN:   INB       0FDH           ; volver al 8085
                        ;
                                  END                      ; fin del segmento SYS1
```

Figura 7.12b: Segmento del sistema #1.

Comentarios: El programa-ejemplo muestra cómo utilizar las operaciones de desplazamiento para convertir un número de representación binaria a representación hexadecimal. La rutina HEXOUT tiene como entrada un número de 16 bits en el registro DX. La salida en su representación hexadecimal sobre el dispositivo de salida de consola, y la HEXIN entra un número hexadecimal del teclado y devuelve su representación binaria en el registro DX.

Se introducen dos segmentos en este ejemplo. El programa principal se ubica en la posición estándar 01000H. El segmento SYS1 está en la posición 900H y contiene las rutinas del sistema HEXOUT, HEXIN, SPCOUT, CRLF y MESOUT. Son todas ellas en realidad rutinas de E/S a un nivel ligeramente más alto que las del segmento I/O. Llaman a rutinas del segmento I/O, de forma que dicho segmento también debe estar presente en su posición habitual 800H. Nótese que MESOUT es casi igual que en el ejemplo anterior, aunque ahora se ha diseñado para ser llamada de otro segmento distinto al que está, volviendo con la instrucción RETS. Puede ser útil añadir este segmento a la orden GO.

El programa principal introduce dos números (en notación hexadecimal) y saca la suma, la diferencia y el producto.

La rutina HEXOUT graba el número de 16 bits de 4 en 4 bits y utiliza un *código inteligente*[3], para convertir cada grupo de 4 bits en el código ASCII, para su representación hexadecimal correspondiente. Más concretamente, el número de 16 bits se rota

[3] Un código inteligente es un código que utiliza ideas más difíciles de comprender para conseguir que la máquina funcione más económicamente.

a la izquierda cuatro posiciones cada vez con la siguiente secuencia de dos instrucciones:

```
HLOOP:  MOVBI   CL,4    ; contador igual a 4
                        ; para cada uno
        ROL     DX,CL   ; desplazar
                        ; a la izquierda
```

A continuación se lleva al acumulador, donde se enmascaran los 4 bits más bajos haciendo el AÑD del acumulador con la posición 000FH. La siguiente secuencia de código inteligente sirve para convertir en ASCII.

```
        DAA             ; sumar 6 si
                        ; A-F
        ADDBI   AL,0F0H ; generar un
                        ; acarreo si
                        ; A-F
        ADCBI   AL,040H ; aquí aparece
                        ; el ASCII
```

Puesto que hemos decidido utilizar un código inteligente, deberíamos explicar cómo funciona. Aunque tal código puede funcionar más rápido y ocupar menos espacio, puede costar más en total si se cuenta el tiempo empleado en su desarrollo, y el tiempo necesario para que un programador nuevo lo aprenda cada vez que necesite mantenimiento. Veamos cómo funciona nuestro ejemplo particular en dos casos:

1. Si el número está comprendido entre el 0 y 9, la DAA no cambia nada, de forma que la suma de F0H no hace nada salvo cambiar el cuarteto superior al valor 15. En este caso no hay acarreo, y la instrucción siguiente suma 4 al cùarteto superior, resultando 3.
(15+4=19=16+3=acarreo+3).
2. Si el número está comprendido entre 10 y 15, la DAA suma 6, haciendo que el cuarteto superior pase a valer 1. Con esto el número «parece» en representación decimal. Al sumarle F0H, el cuarteto superior toma el valor 0, y el acarreo se pone a 1. Así, el acumulador mantiene el valor original menos 10. La última instrucción suma el acarreo y pone 4 en el cuarteto superior. Esto es lo mismo que sumar 41H, que es el código ASCII para la A, haciendo corresponder estos valores con los símbolos de la A a la F. La tabla 7.1 muestra lo que sucede:

número original (hex.)	0A	0B	0C	0D	0E	0F
después de la 1.ª instrucción	10	11	12	13	14	15
después de la 2.ª instrucción	00	01	02	03	04	05
después de la 3.ª instrucción	41	42	43	44	45	46
símbolo correspondiente	A	B	C	D	E	F

Tabla 7.1: Seguimiento de un código inteligente.

Después de este código inteligente, el dígito se envía a la salida y el programa vuelve al comienzo del bucle para tratar al siguiente dígito.

Como las tres rutinas SPCOUT, CRLF y MESOUT se usan con mucha frecuencia, las hemos incluido en este segmento. Invitamos al lector a introducir más rutinas en este segmento.

Instrucciones de pila

Una de las misiones de la pila del sistema es la de salvaguardar (conservar) datos (la otra es la de salvaguardar las direcciones de vuelta de las llamadas a subrutinas). Las instrucciones de salvaguarda de datos son las siguientes:

PUSH	Introducir
POP	Extraer
PUSHF	Introducir indicaciones
POPF	Extraer indicaciones

La figura 7.13 muestra el ejemplo de un programa que utiliza las instrucciones PUSH y POP para salvaguardar y restaurar los contenidos de los registros del 8088.

```
                                ; Ejemplo 9: (Instrucciones sobre la pila)
                                ; volcado de todos los registros del 8088
                                ;
0000'                                   ASEG
00A0                            SYS2    EQU     0A0H            ; segmento del sistema #2
0000                            DISR    EQU     0000H           ; visualizar registros
0000                            ENTER:
0000    B8 0001                         MOVI    AX,1            ; cargar AX, BX, CX y DX
0003    BB 0002                         MOVI    BX,2
0006    B9 0003                         MOVI    CX,3
0009    BA 0004                         MOVI    CX,4
000C    9A 0000                         CALL    DISR,SYS2       ; visualizar todos los registros
000F    00A0
0011    E4 FD                           INB     0FDH            ; vuelta al 8088
                                ;
                                        END                     ; Fin del segmento código
```

Figura 7.13a: Ejemplo 9: Volcado de todos los registros del 8088.

```
); Segmento del sistema n.º 2
                                ;
                                ; Este segmento contiene una rutina de visualización
                                ; de los contenidos de todos los registros del 8088.
                                ; Hay también una rutina local de soporte.
                                ;
                                ; La rutina principal es:
                                ;
                                ;    DISR:   Visualiza el contenido de los
                                ;            registros AX, BX, CX y DX como
                                ;            números hexadecimales de 16 bits.
                                ;            Preserva todos los registros.
                                ;
0000'                                   ASEG
0080                            IO      EQU     80H             ; Segmento IO
0026                            CDOUT   EQU     26H             ; desplazamiento de CDOUT
0020                            CDIN    EQU     20H             ; desplazamiento de CDIN
0090                            SYS1    EQU     90H             ; segmento SYS1
0000                            HEXOUT  EQU     00H             ; desplazamiento de HEXOUT
004B                            SPCOUT  EQU     4BH             ; desplazamiento de SPCOUT
```

```
0053                      CRLF      EQU     53H             ; desplazamiento de CRLF
0062                      MESOUT    EQU     62H             ; desplazamiento de MESOUT
                          ;
0000                      DISR:                             ; rutina de visualización de registros
                          ;
0000    9C                          PUSHF
0001    50                          PUSH    AX              ; salvaguardar registros EU
0002    53                          PUSH    BX
0003    51                          PUSH    CX
0004    52                          PUSH    DX
0005    56                          PUSH    SI              ; y registros BIU seleccionados
0006    55                          PUSH    BP
0007    1E                          PUSH    DS
                          ;
0008    06                          PUSH    ES              ; salvaguardarlos de nuevo
0009    1E                          PUSH    DS              ; para la visualización
000A    16                          PUSH    SS
000B    0E                          PUSH    CS
000C    50                          PUSH    AX              (un avisador para IP)
000D    54                          PUSH    SP
000E    55                          PUSH    BP
000F    56                          PUSH    SI
0010    57                          PUSH    DI
0011    9C                          PUSHF
0012    52                          PUSH    DX
0013    51                          PUSH    CX
0014    53                          PUSH    BX
0015    50                          PUSH    AX
                          ;
0016    8B EC                       MOV     BP,SP           ; fijar algunos de ellos
                          ;
0018    8B 46 2E                    MOV     AX,[BP+46]      ; el CS
001B    89 46 14                    MOV     [BP+20],AX
                          ;
001E    8B 46 2C                    MOV     AX,[BP+44]      ; el IP
0021    89 46 12                    MOV     [BP+18],AX
                          ;
0024    8B C5                       MOV     AX,BP           ; el SP
0026    05 0030                     ADDI    AX,48
0029    89 46 10                    MOV     [BP+16],AX
                          ;
002C    8C C8                       MOV     AX,CS           ; permite que el segmento de datos sea el
002E    8E D8                       MOV     DS,AX           ; mismo que el segmento de código
                          ;
0030    9A 0053                     CALL    CRLF,SYS1       ; <CR><LF>
0033    0090
0035    BE 00BD                     MOVI    SI,MESSAX       ; registro AX
0038    E8 0066                     CALL    REGOUT
003B    BE 00C2                     MOVI    SI,MESSBX       ; registro BX
003E    E8 0060                     CALL    REGOUT
0041    BE 00C7                     MOVI    SI,MESSCX       ; registro CX
0044    E8 005A                     CALL    REGOUT
0047    BE 00CC                     MOVI    SI,MESSDX       ; registro DX
004A    E8 0054                     CALL    REGOUT
004D    BE 00D1                     MOVI    SI,MESSFG       ; registro de indicadores
0050    E8 004E                     CALL    REGOUT
                          ;
0053    9A 0053                     CALL    CRLF,SYS1       ; <CR><LF>
0056    0090
0058    BE 00D9                     MOVI    SI,MESSDI       ; registro DI
005B    E8 0043                     CALL    REGOUT
005E    BE 00DE                     MOVI    SI,MESSI        ; registro SI
0061    E8 003D                     CALL    REGOUT
0064    BE 00E3                     MOVI    SI,MESSBP       ; registro BP
0067    E8 0037                     CALL    REGOUT
006A    BE 00E8                     MOVI    SI,MESSSP       ; registro SP
006D    E8 0031                     CALL    REGOUT
0070    BE 00ED                     MOVI    SI,MESSIP       ; registro IP
0073    E8 002B                     CALL    REGOUT
                          ;
0076    9A 0053                     CALL    CRLF,SYS1       ; <CR> <LF>
0079    0090
007B    BE 00F2                     MOVI    SI,MESSCS       ; registro CS
007E    E8 0020                     CALL    REGOUT
0081    BE 00F7                     MOVI    SI,MESSSS       ; registro SS
0084    E8 001A                     CALL    REGOUT
0087    BE 00FC                     MOVI    SI,MESSDS       ; registro DS
008A    E8 0014                     CALL    REGOUT
008D    BE 0101                     MOVI    SI,MESSES       ; registro ES
0090    E8 000E                     CALL    REGOUT
```

```
                         ;
0093    9A 0053                  CALL    CRLF,SYS1               ; <CR><LF>
0096    0090
                         ;
0098    1F                       POP     DS                      ; restaurar registros
0099    5D                       POP     BP
009A    5E                       POP     SI
009B    5A                       POP     DX
009C    59                       POP     CX
009D    5B                       POP     BX
009E    58                       POP     AX
009F    9D                       POPF
00A0    CB                       RETS                            ; Fin de DISP
                         ;
00A1                     REGOUT:                                 ; rutina de visualización de un registro
00A1    9A 0062                  CALL    MESOUT,SYS1             ; nombre del registro
00A4    0090
00A6    8B EC                    MOV     BP,SP                   ; apunta a la pila
00A8    8B 56 02                 MOV     DX,2[BP]                ; obtener dato de la pila
00AB    9A 0000                  CALL    HEXOUT,SYS1             ; salida hexadecimal
00AE    0090
00B0    9A 004B                  CALL    SPCOUT,SYS1             ; un par de espacios
00B3    0090
00B5    9A 004B                  CALL    SPCOUT,SYS1
00B8    0090
00BA    C2 0002                  RET     2                       ; vuelta y ajustar pila
                         ;
00BD    41 58 3D 20      MESSAX: DB      'AX= ',0
00C1    00
00C2    42 58 3D 20      MESSBX: DB      'BX= ',0
00C6    00
00C7    43 58 3D 20      MESSCX: DB      'CX= ',0
00CB    00
00CC    44 58 3D 20      MESSDX: DB      'DX= ',0
00D0    00
00D1    46 4C 41 47      MESSFG: DB      'FLAGS= ',0
00D5    53 3D 20 00
00D9    44 49 3D 20      MESSDI: DB      'DI= ',0
00DD    00
00DE    53 49 3D 20      MESSSI: DB      'SI= ',0
00E2    00
00E3    42 50 3D 20      MESSBP: DB      'BP= ',0
00E7    00
00E8    53 50 3D 20      MESSSP: DB      'SP= ',0
00EC    00
00ED    49 50 3D 20      MESSIP: DB      'IP= ',0
00F1    00
00F2    43 53 3D 20      MESSCS: DB      'CS= ',0
00F6    00
00F7    53 53 3D 20      MESSSS: DB      'SS= ',0
00FB    00
00FC    44 53 3D 20      MESSDS: DB      'DS= ',0
0100    00
0101    45 53 3D 20      MESSES: DB      'ES= ',0
0105    00
                         ;
                                 END                             ; Fin del sistema n.º 2
```

Figura 7.13b: Segmento del sistema n.º 2.

Comentarios: El ejemplo muestra cómo utilizar la pila para preservar el contenido de los registros durante las llamadas a subrutinas. Ilustra también cómo usar la pila para pasar valores a las subrutinas.

Se introducen dos segmentos, el del programa principal que se ubica en la posición estándar 1000H, y SYS2, un segundo segmento del sistema, que se ubica en A00H. Los segmentos IO y SYS1 deberían cargarse en las posiciones usuales. En este punto es recomendable que se asegure que ha introducido todos estos segmentos en el área del orden GO (de 100H a 1000H).

El programa principal carga los registros AX, BX, CX y DX

con ciertos números y llama a la rutina de visualización de registros DISR en el segmento SYS2.

Esta rutina comienza salvando (conservando) varios registros introduciéndolos en la pila, y luego salva todos los registros en orden inverso. Esta última salvaguarda pone una copia de los registros en la pila mediante la rutina REGOUT. La rutina REGOUT se llama una vez por cada registro. Esta rutina necesita que el registro SI apunte al mensaje que *rotula* el registro a visualizar. La rutina «engulle» una cantidad de 16 bits de la pila, que es el contenido de este registro, tal como estaba en el programa principal.

La rutina REGOUT trabaja de la siguiente manera: Se envía primero el rótulo del registro. A continuación se coloca el contenido del puntero de pila en el puntero de base (BP), y el contenido del registro se extrae de la pila desde el otro lado de la dirección de vuelta (retorno) y se coloca en el registro DX. Se llama entonces a la rutina HEXOUT para que visualice el número y se saltan dos espacios.

La vuelta de REGOUT se hace con la instrucción RET 2 que vuelve y ajusta el puntero de pila a 2 octetos para saltar el dato que se ha pasado a la rutina.

Después de llamar a REGOUT desde DISR, se envía a la salida un ⟨CR⟩ ⟨LF⟩, y se restauran los registros extrayendo sus valores de la pila en el orden inverso a como se salvaguardaron.

Notemos que la pila se utiliza para datos (datos a largo término) y para llamadas a subrutinas. Esto requiere un planteamiento especial y un poco de unidad en las maniobras.

Control del procesador

Hay varias instrucciones de control de la CPU, sea a ella sola, o en conjunción con otros procesadores. Las instrucciones de este grupo incluyen:

NOP	No operación
HLT	Parada
WAIT	Espera
LOCK	Bloquea
ESC	Escape

La figura siguiente muestra un ejemplo que utiliza dos de estas instrucciones, exactamente las HLT y NOP.

```
                        ; Ejemplo 10: (Control del procesador)
                        ;
                        ; NOP y STOP
                        ;
0000'                           ASEG
00A0                    SYS2    EQU     0A0H            ; segmento del sistema n.º 2
0000                    DISR    EQU     0000H           ; desplazamiento de DISR
                        ;
0000                    ENTER:
0000    9A 0000                 CALL    DISR,SYS2       ; visualizar registros
0003    00A0
0005    90                      NOP                     ; no operación
0006    9A 0000                 CALL    DISR,SYS2       ; visualizar registros
0009    00A0
```

```
000B    F4              HLT                     ; parada
000C    9A 0000         CALL    DISR,SYS2       ; no se debería llegar tan lejos
000F    00A0
0011    E4 FD           INB     0FDH            ; vuelta al 8088
                ;
                        END                     ; Fin del segmento código
```

Figura 7.14: Ejemplo 10: NOP y STOP.

Comentarios: El ejemplo muestra lo que sucede cuando se utiliza la instrucción NOP y se para la CPU con la instrucción HLT. La rutina DISR controla el proceso a lo largo del código.

La rutina se puede cargar en la posición estándar 1000H. Necesita además de los segmentos IO, SYS1 y SYS2 en sus posiciones usuales. (¿Ha preparado la orden GO?)

INSTRUCCIONES DE INTERRUPCION

Hay varias instrucciones de ayuda a la estructura de interrupciones del 8086/8088. Son las siguientes:

STI	Poner a 1 el indicador de interrupción
CLI	Borrar el indicador de interrupción
INT	Interrupción
INTO	Interrupción por capacidad excedida (desbordamiento)
IRET	Interrupción por vuelta (retorno)

```
; Ejemplo 11: (Interrupciones)
; PAQUETE DE DEPURACION
;
        ASEG
SPACE   EQU     20H             ; código ASCII para SPACE
CR      EQU     0DH             ; código ASCII para RETURN
LF      EQU     0AH             ; código ASCII para LINEFEED
IO      EQU     80H             ; Segmento IO
CDOUT   EQU     26H             ; salida por consola
CDIN    EQU     20H             ; entrada por consola
SYS1    EQU     90H             ; segmento del sistema n.º 1
HEXOUT  EQU     00H             ; salida hexadecimal
HEXIN   EQU     IBH             ; entrada hexadecimal
SPCOUT  EQU     4BH             ; sacar un espacio
CRLF    EQU     53H             ; sacar <CR><LF>
MESOUT  EQU     62H             ; sacar un mensaje
;
; ******************************************************************************
; El siguiente código inicializa los vectores de interrupción.
;
ENTER                           ; punto de entrada inicial
        $CS
        POP     RETADD          ; salvaguardar direcciones de vuelta
        $CS
        POP     RETADD+2
        ;
        PUSHF                   ; y restaurar la pila
        $CS
        PUSH    RETADD+2
        $CS
        PUSH    RETADD
        PUSH    AX              ; salvaguardar registros de trabajo
        PUSH    BX
        PUSH    CX
        PUSH    DX
        PUSH    SI
        PUSH    BP
        PUSH    DS
;
```

```
        MOVI    AX,0                    ; apuntar al
        MOV     DS,AX                   ; comienzo de memoria
        MOV     AX,CS                   ; y obtener este segmento código
;
                                        ; preparar interrupciones
        MOVI    04H,IBRK                ; desplazamiento
        MOV     0GH,AX                  ; dirección del segmento
        MOVI    0CH,JBRK                ; desplazamiento
        MOV     0EH,AX                  ; dirección de este segmento código
;
        MOV     DS,AX
;
        MOVI    SI,MESS0                ; mensaje de bienvenida
        CALL    MESOUT,SYS1
;
        JMP     JBRK                    ; saltar a restaurar y
                                        ; visualizar registros
;
; ***********************************************************************
; El código siguiente procesa la interrupción por punto de ruptura.
;
JBRK:
        PUSH    AX                      ; salvaguardar los registros estándar
        PUSH    BX
        PUSH    CX
        PUSH    DX
        PUSH    SI
        PUSH    BP
        PUSH    DS
;
        MOV     BP,SP                   ; BP apunta a la pila
        $CS
        MOVB    AL,XBREAK               ; octeto de código que falta
        DEC     [BP+0EH]                ; aumentar IP
        LDS     BX,[BP+0EH]             ; apuntar al código
        MOVB    [BX],AL                 ; introducir el octeto de código que falta
;
JRET:   POP     DS                      ; restaurar registros estándar
        POP     BP
        POP     SI
        POP     DX
        POP     CX
        POP     BX
        POP     AX
        JMP     IBRK                    ; saltar a visualizar registros
;
; ***********************************************************************
; El código siguiente sirve una interrupción paso a paso.
; Visualiza todos los registros del 8088, la pila,
; y la cola de instrucciones. Permite modificar los registros,
; definir puntos de ruptura y "GO" en cualquier sitio.
;
IBRK:                                   ; procedimiento puntos de ruptura
;
        PUSH    AX                      ; salvaguardar registros estándar
        PUSH    BX
        PUSH    CX
        PUSH    DX
        PUSH    SI
        PUSH    BP
        PUSH    DS
;                                       ; salvaguardar de nuevo para visualización
        PUSH    AX                      ; (avisadores para la pila)
        PUSH    AX
        PUSH    AX
        PUSH    AX
        PUSH    ES
        PUSH    DS
        PUSH    SS
        PUSH    CS
        PUSH    AX                      ; (avisador para IP)
        PUSH    SP
        PUSH    BP
        PUSH    SI
        PUSH    DI
        PUSH    AX                      ; ((avisador para los indicadores)
        PUSH    DX
        PUSH    CX
        PUSH    BX
        PUSH    AX
```

```
;
        MOV     BP,SP                   ; fijar algunos de ellos
;
        MOV     AX,[BP+54]              ; los indicadores
        MOV     [BP+8],AX
;
        MOV     AX,[BP+52]              ; el CS
        MOV     [BP+20],AX
;
        MOV     AX,[BP+50]              ; el IP
        MOV     [BP+18],AX
;
        MOV     AX,BP                   ; el SP
        ADDI    AX,56
        MOV     [BP+16],AX
;
        MOV     AX,[BP+56]              ; la pila
        MOV     [BP+28],AX
        MOV     AX,[BP+58]
        MOV     [BP+30],AX
        MOV     AX,[BP+60]
        MOV     [BP+32],AX
        MOV     AX,[BP+62]
        MOV     [BP+34],AX
;
IBRK0:  MOV     AX,CS                   ; permite que el segmento de datos coincida
        MOV     DS,AX                   ; con el segmento de código
;
BODY:                                   ; visualizalos
        CALL    CRLF,SYS1               ; <CR><LF>
        MOVI    SI,MESSAX               ; registro AX
        CALL    REGOUT
        MOVI    SI,MESSBX               ; registro BX
        CALL    REGOUT
        MOVI    SI,MESSCX               ; registro CX
        CALL    REGOUT
        MOVI    SI,MESSDX               ; registro DX
        CALL    REGOUT
        MOVI    SI,MESSFG               ; registro de indicadores
        CALL    REGOUT
;
        CALL    CRLF,SYS1               ; <CR><LF>
        MOVI    SI,MESSDI               ; registro DI
        CALL    REGOUT
        MOVI    SI,MESSSI               ; registro SI
        CALL    REGOUT
        MOVI    SI,MESSBP               ; registro BP
        CALL    REGOUT
        MOVI    SI,MESSSP               ; registro SP
        CALL    REGOUT
        MOVI    SI,MESSIP               ; registro IP
        CALL    REGOUT
;
        CALL    CRLF,SYS1               ; <CR><LF>
        MOVI    SI,MESSCS               ; registro CS
        CALL    REGOUT
        MOVI    SI,MESSSS               ; registro SS
        CALL    REGOUT
        MOVI    SI,MESSDS               ; registro DS
        CALL    REGOUT
        MOVI    SI,MESSES               ; registro ES
        CALL    REGOUT
;
        CALL    CRLF,SYS1               ; <CR><LF>
        MOVI    SI,MESSS0               ; pila + 0
        CALL    REGOUT
        MOVI    SI,MESSS1               ; pila + 1
        CALL    REGOUT
        MOVI    SI,MESSS2               ; pila + 2
        CALL    REGOUT
        MOVI    SI,MESSS3               ; pila + 3
        CALL    REGOUT
;
        CALL    CRLF,SYS1               ; <CR><LF>
;
;                                       visualizar cola
        MOVI    SI,MESSQU
        CALL    MESOUT,SYS1
        MOVI    SI,0
        CALL    DQUEUE
```

```
        MOVI    SI,1
        CALL    DQUEUE
        MOVI    SI,2
        CALL    DQUEUE
        MOVI    SI,3
        CALL    DQUEUE
        MOVI    SI,4
        CALL    DQUEUE
        MOVI    SI,5
        CALL    DQUEUE
;
        CALL    CRLF,SYS1           ; <CR><LF>
;
IBRK1:  MOV     AX,CS               ; dar a DS y BP
        MOV     DS,AX               ; sus valores
        MOV     BP,SP               ; estándar
;
        CALL    CDIN,IO             ; esperar la clave
;
        CMPBI   AL,'R'              ; preguntar por las órdenes:
        JZ      GORET               ; Vuelta de subrutina
        CMPBI   AL,'E'
        JZ      EXIT                ; Salir del depurador
        CMPBI   AL,'B'
        JZ      SETBRK              ; Definir punto de ruptura
        CMPBI   AL,'G'
        JZ      GO                  ; Ir a una dirección especificada
        CMPBI   AL,'C'
        JZ      CONT                ; Continuar (no paso a paso)
        CMPBI   AL,'S'
        JZ      SETREG              ; Preparar el contenido del registro
        CMPBI   AL,SPACE
        JZ      GOSS                ; continuar paso a paso
        JMP     IBRK1               ; realizar bucle si no ocurre nada de lo anterior
;
GORET:                              ; Vuelta de subrutina
        LDS     BX,[BP+ØEH]         ; apuntar a la dirección de vuelta
        MOV     BX,[BP+14H]
        MOVBI   AL,ØCCH             ; poner el punto de ruptura
        XCHGB   [BX],AL             ; reemplazando al octeto de código
        $CS
        MOVB    XBREAK,AL           ; salvaguardar el octeto de código para más tarde
        JMP     CONT                ; y desviarse al comienzo
;
EXIT:   JMPL    RETADD              ; salto indirecto largo de vuelta
;
SETBRK:                             ; Procedimiento de activación de puntos de ruptura
        CALL    CRLF,SYS1
        MOVI    SI,MESS2            ; mensaje del desplazamiento
        CALL    MESOUT,SYS1
        CALL    HEXIN,SYS1          ; obtener el desplazamiento
        LDS     BX,[BP+ØEH]         ; apuntar a la posición del punto de ruptura
        MOV     BX,DX
        MOVBI   AL,ØCCH             ; poner el punto de ruptura
        XCHGB   [BX],AL             ; reemplazando al octeto de código
        $CS
        MOVB    XBREAK,AL           ; salvaguardar el octeto de código para más tarde
        JMP     IBRK1               ; ¿más instrucciones?
;
GO:                                 ; Ir a la dirección especificada
        CALL    CRLF,SYS1
        MOVI    SI,MESS1            ; obtener segmento
        CALL    MESOUT,SYS1
        CALL    HEXIN,SYS1
        MOV     [BP+1ØH],DX
        CALL    CRLF,SYS1
        MOVI    SI,MESS2            ; obtener desplazamiento
        CALL    MESOUT,SYS1
        CALL    HEXIN,SYS1
        MOV     [BP+ØEH],DX
CONT:   ANDI    [BP+12H],ØFEFFH     ; borrar el paso a paso
        JMP     IBRK3               ; dejar el depurador
;
GOSS:                               ; paso a paso
        DRI     [BP+12H],1ØØH       ; activar el paso a paso
        JMP     IBRK3               ; dejar el depurador
;
SETREG:                             ; Procedimiento de inicialización de registros
        CALL    CRLF,SYS1
        MOVI    SI,MESS3            ; Obtener registro
```

```
                CALL    MESOUT,SYS1
                CALL    CDIN,IO             ; primera letra en CH
                MOVB    AH,AL
                CALL    CDOUT,IO
                MOVB    CH,AH
                CALL    CDIN,IO             ; segunda letra en CL
                MOVB    AH,AL
                CALL    CDOUT,IO
                MOVB    CL,AH
;
                PUSH    CX                  ; Salvaguardar nombre de registro
                CALL    CRLF,SYS1
                MOVI    SI,MESS4            ; Obtener contenido nuevo
                CALL    MESOUT,SYS1
                CALL    HEXIN,SYS1
                POP     AX                  ; AX contiene el nombre del registro
;
TSTAX:          CMPI    AX,'A'*100H+'X'     ; ¿AX?
                JNZ     TSTBX
                MOVI    SI,0CH              ; desplazamiento de AX
                JMPS    LDREG
TSTBX:          CMPI    AX,'B'*100H+'X'     ; ¿BX?
                JNZ     TSTCX
                MOVI    SI,0AH              ; desplazamiento de BX
                JMPS    LDREG
TSTCX:          CMPI    AX,'C'*100H+'X'     ; ¿CX?
                JNZ     TSTDX
                MOVI    SI,08H              ; desplazamiento de CX
                JMPS    LDREG
TSTDX:          CMPI    AX,'D'*100H+'X'     ; ¿DX?
                JNZ     TSTSI
                MOVI    SI,06H              ; desplazamiento de DX
                JMPS    LDREG
TSTSI:          CMPI    AX,'S'*100H+'I'     ; ¿SI?
                JNZ     TSTDI
                MOVI    SI,04H              ; desplazamiento de SI
                JMPS    LDREG
TSTDI:          CMPI    AX,'D'*100H+'I'     ; ¿DI?
                JNZ     TSTBP
                MOV     DI,DX               ; hacerlo directamente
                JMP     JRET
TSTBP:          CMPI    AX,'B'*100H+'P'     ; ¿BP?
                JNZ     TSTSP
                MOVI    SI,02H              ; desplazamiento para BP
                JMPS    LDREG
TSTSP:          CMPI    AX,'S'*100H+'P'     ; ¿SP?
                JNZ     TSTFL
                JMP     IBRK1               ; no modificar SP
TSTFL:          CMPI    AX,'F'*100H+'L'     ; ¿FL?
                JNZ     TSTIP
                MOVI    SI,12H              ; desplazamiento para los indicadores
                JMPS    LDREG
TSTIP:          CMPI    AX,'I'*100H+'P'     ; ¿IP?
                JNZ     TSTCS
                MOVI    SI,0EH              ; desplazamiento para IP
                JMPS    LDREG
TSTCS:          CMPI    AX,'C'*100H+'S'     ; ¿CS?
                JNZ     TSTDS
                MOVI    SI,10H              ; desplazamiento para CS
                JMPS    LDREG
TSTDS:          CMPI    AX,'D'*100H+'S'     ; ¿DS?
                JNZ     TSTES
                MOVI    SI,00H              ; desplazamiento para DS
                JMPS    LDREG
TSTES:          CMPI    AX,'E'*100H+'S'     ; ¿ES?
                JNZ     TSTSS
                MOV     ES,DX               ; hacerlo directamente
                JMPS    LDREG
TSTSS:          CMPI    AX,'S'*100H+'S'     ; ¿SS?
                JNZ     TSTEND              ; no modificar SS
TSTEND:         JMP     IBRK1               ; ninguno de arriba
;
LDREG:          MOV     [BP][SI],DX
                JMP     JRET
;
IBRK3:          POP     DS                  ; restaurar registros
                POP     BP
                POP     SI
                POP     DX
                POP     CX
```

```
         POP     BX
         POP     AX
         IRET                    ; Fin de IBRK
;
; ****************************************************************************
; Lo que sigue son rutinas usadas por IBRK
;
REGOUT:                          ; rutina de visualización de un registro
         CALL    MESOUT,SYS1     ; nombre de registro
         MOV     BP,SP           ; apuntar a la pila
         MOV     DX,2[BP]        ; obtener datos de la pila
         CALL    HEXOUT,SYS1     ; salida hexadecimal
         CALL    SPCOUT,SYS1     ; un par de espacios
         CALL    SPCOUT,SYS1
         RET     2               ; volver y ajustar la pila
;
DQUEUE:                          ; visualizar cola
         MOV     BX,[BP+18]
         MOV     AX,[BP+20]      ; segmento de código viejo
         MOV     DS,AX           ; a segmento de datos
         MOVBI   DH,0            ; obtener octeto de instrucción
         MOVB    DL,[BX][SI]
         MOV     AX,CS           ; nuestro segmento código
         MOV     DS,AX           ; volver otra vez
         CALL    HEXOUT,SYS1     ; mostrar el valor
         CALL    SPCOUT,SYS1     ; saltar espacio
         RET
;
; ****************************************************************************
; Lo que sigue son posiciones de almacenamiento para las variables
;
XBREAK:  DB      0
RETADD:  DW      0               ; salvaguardar desplazamiento
         DW      0               ; salvaguardar segmento de código
                                 ;
; ****************************************************************************
; Lo que sigue son mensajes
;
MESS0:   DB      'Debus Program'
         DB      CR,LF,'R     = return from suroutine'
         DB      CR,LF,'B     = set to break point'
         DB      CR,LF,'E     = exit debus prosram'
         DB      CR,LF,'S     = set resister'
         DB      CR,LF,'<SP>  = sinsle step'
         DB      CR,LF,'G     = so to address'
         DB      CR,LF,0
MESS1:   DB      'Code Segment: ',0
MESS2:   DB      'Offset:',0
MESS3:   DB      'Resister: ',0
MESS4:   DB      'Contents: ',0
MESSAX:  DB      'AX= ',0
MESSBX:  DB      'BX= ',0
MESSCX:  DB      'CX= ',0
MESSDX:  DB      'DX= ',0
MESSFG:  DB      'FLAGS= ',0
MESSDI:  DB      'DI= ',0
MESSSI:  DB      'SI= ',0
MESSBP:  DB      'BP= ',0
MESSSP:  DB      'SP= ',0
MESSIP:  DB      'IP= ',0
MESSCS:  DB      'CS= ',0
MESSSS:  DB      'SS= ',0
MESSDS:  DB      'DS= ',0
MESSES:  DB      'ES= ',0
MESSS0:  DB      'STACK+0= ',0
MESSS1:  DB      'STACK+2= ',0
MESSS2:  DB      'STACK+4= ',0
MESSS3:  DB      'STACK+6= ',0
MESSQU:  DB      'QUEUE = ',0
;
; ****************************************************************************
         END
; ****************************************************************************
```

Figura 7.15: Ejemplo 11: Paquete de depuración (sólo código fuente).

Comentarios: Esta es la «traca» final. Es algo más largo que otros ejemplos y por lo tanto, en interés del espacio, sólo hemos

incluido el código fuente. El programa permite depurar cualquier código máquina del 8088, incluyendo el de estos mismos ejemplos. Muestra el contenido de todos los registros y de la pila en cada paso del programa, y là cola de instrucciones. Permite también varias opciones, incluyendo la de ejecución paso a paso del código 8088, y la de introducir puntos de ruptura.

El programa no es un paquete completo DEBUG de depuración, porque no se puede volcar o cargar las posiciones de memoria, ni visualizar el código ensamblador. Sería un buen ejercicio el añadir algunas de estas posibilidades.

Este segmento se puede ubicar en la dirección 1000H y necesita los segmentos IO, y SYS1 en las posiciones estándar. Le recomendamos que incluya también este segmento en la orden GO; puede llamar al total DEBUG.COM.

El programa comienza inicializando varias posiciones de la parte más baja de memoria, que se utilizan como vectores de interrupción para las posibilidades de ir *paso a paso* y de añadir *puntos de ruptura.* En el capítulo 3 se vio la estructura de interrupciones de la CPU 8086/8088. En nuestro ejemplo, los vectores de interrupción se posicionan en las direcciones absolutas 04h, 06h, 0Ch y 0Eh. Las dos primeras son para la interrupción de avance *paso a paso,* de tipo 1, y las dos siguientes para la interrupción de *octeto simple,* de tipo 3. Recuérdese que el «tipo» de interrupción se refiere a la posición de su *vector de interrupción* (el tipo por 4) que contiene un puntero (desplazamiento y número de segmento) a la correspondiente rutina de servicio. Cuidado: Vigile las vueltas al 8085 vía la instrucción INB 0FDH. El CP/M utiliza algunas de estas interrupciones en sus operaciones. Tendrá que recargar si quiere volver a CP/M.

A continuación el programa restaura algunos de los registros y va al cuerpo principal, que visualiza los registros, la pila y la cola y permite un cierto número de opciones. Entre ellas:

B	Activar punto de ruptura
R	Vuelta de subrutina
E	Salida de DEBUG
S	Activar registro
espacio	Paso simple
G	Ir a una posición especificada

Repasemos estas órdenes, comenzando por la de paso simple. Para activarla basta con introducir un espacio. El resultado será una nueva visualización de los registros, la pila y la cola de instrucciones. Se debe ver cómo el IP avanza a la instrucción siguiente y cómo la cola de instrucciones sube una posición. Se debe ir al lenguaje ensamblador para ver cómo se implementa esta orden.

La operación lógica OR se usa para poner a 1 el bit de interrupción (trap) en la *copia* de los indicadores de la pila del sistema. La instrucción IRET siguiente lleva esta copia de los indicadores a los indicadores reales. En el modo paso a paso se

genera una interrupción después de cada instrucción, lo que hace que la CPU llame a la rutina IBRK que muestra la visualización. La CPU sabe dónde está IBRK porque habremos cargado su número de segmento y su desplazamiento en el vector de interrupción número 1 al final de memoria.

La orden B permite colocar una instrucción de interrupción (de 1 octeto) en cualquier lugar dentro del segmento de código en curso (como indica el valor de CS en la visualización). Para utilizar esta orden basta con introducir B. El programa pedirá el desplazamiento. Introduzca la dirección de desplazamiento de la posición en la que quiera insertar el punto de ruptura, seguido de una vuelta de carro. El programa introduce el punto de ruptura y espera nuevas órdenes. El programa en realidad reemplaza el octeto de código de la posición dada por el desplazamiento por una instrucción de interrupción tipo 3 de 1 solo octeto, y salvaguarda el octeto de código en una posición especial para poder tratar la ruptura cuando se produzca. Cuando el procesador llega al punto de ruptura, vectoriza JBRK, que hace que el octeto original del código se reemplace, y salta a IBRK para una visualización de las órdenes siguientes.

La orden R se debe usar sólo cuando la dirección de vuelta de una subrutina aparezca en el elemento superior de la pila (cosa que se puede ver en la visualización). Esto sucede normalmente cuando se avanza paso a paso a través del código y se desea saltar una subrutina. Se avanzaría paso a paso hasta la llamada a la subrutina, que colocaría la dirección de vuelta en la pila del sistema. Esta instrucción activa automáticamente un punto de ruptura de una forma similar a como lo hace la orden B.

La orden E está pensada para utilizar con un monitor o un sistema en disco (DOS) que llamara la DEBUG desde un segmento distinto (CALL con dos operandos). La inicialización establece la dirección de vuelta en una posición especial que utiliza la orden E para saltar a ella (indirectamente). Si no se ha llamado al programa DEBUG de esta forma, ¡la orden *no* funciona!

La orden S permite poner a 1 cualquier registro salvo el SP y el SS. El poner a 1 estos registros liaría el programa DEBUG. Para llamar esta orden basta con introducir S. Como respuesta el sistema pide un registro. Cuando el usuario contesta, pide el contenido. Este se debe entrar en hexadecimal, seguido de una vuelta de carro. A continuación aparece visualizado el registro con el nuevo valor introducido.

La orden G permite saltar a cualquier dirección. Pregunte el valor del segmento de código y del desplazamiento dentro del segmento.

El lector debería intentar implementar este programa, y utilizarlo para trabajar con los otros programas-ejemplo. ¡Incluso podría utilizarlo para trabajar consigo mismo!

CONCLUSIONES

En este capítulo se ha intentado que el lector vea al 8086/8088 desde el punto de vista del programador de lenguaje ensamblador. Esperemos que estos ejemplos software le hayan proporcio-

nado una mejor comprensión de cómo se puede utilizar eficazmente el juego de instrucciones del 8086/8088.

Todos estos programas se desarrollaron y probaron sobre una Placa Procesadora Dual Godbout 8085/8088. Todos ellos funcionan y generan resultados que se pueden ver y comprobar. Confiamos que le gusten.

Capítulo 8

Estado actual: Programas y productos del 8086/8088

En este capítulo exploraremos los soportes hardware y software actuales para los procesadores Intel 8086, 8088, 8087 y 8089. Cuando se introdujo por primera vez en 1978 la CPU 8086, no había software ni hardware para el chip 8086, para otra cosa que no fuera temas de investigación y desarrollo. En aquel tiempo, se podía aprender y jugar con el chip, pero era difícil hacerlo trabajar en cuestiones provechosas. Rápidamente, Intel y otras compañías empezaron a incorporar el 8086 (y el 8088) en sus sistemas y a escribir software estándar basado en los juegos de instrucciones usuales de estos dos procesadores. En 1981, IBM introdujo su Computadora Personal basada en la CPU 8088. Esto proporcionó a la CPU 8086/8088 el liderazgo en el campo de las computadoras personales, en oficinas y en escuelas. En estos entornos familiares se puede utilizar en aplicaciones de educación, como procesador de textos, o en procesos generales y de negocios.

EL INICIO. DOS ENFOQUES

Es bien sabido que sin software, una computadora es totalmente inútil, salvo quizás como estufa o como tema de conversación. Hay dos enfoques posibles para equipar una nueva CPU con el software necesario: 1) empezar desde el inicio, desarrollando un software especial para el nuevo sistema, o 2) utilizar un sistema diferente bien desarrollado (el patrón) para escribir el software del nuevo sistema (el objetivo).

El enfoque *patrón-objetivo* fue un éxito con el 8080/8085. Sin embargo, este enfoque es a veces penoso ya que muchas capacidades de gestión, especialmente el tratamiento de ficheros y facilidades de edición, pueden faltar al inicio. Este enfoque fue necesario debido a que muchos diseñadores de software para máquinas basadas en el 8080/8085 no tenían acceso a otras máquinas. Unicamente personas muy expertas usaron minicomputadoras y maxicomputadoras como patrón. Pero la mayoría de expertos no estaban interesados en desarrollar el potencial de los primeros microprocesadores. En vez de ello, se concentraron en un principio en las grandes máquinas. En estos momentos hay un gran interés dentro de la comunidad «informática» por estas máquinas miniatura, desde las compañías «familiares» (tipo «garage») hasta los más grandes gigantes como XEROX, ITT, EXXON e IBM.

El enfoque *patrón-objetivo* requiere los tres pasos siguientes: a) el código fuente se escribe y almacena en el patrón, b) dicho

código se traduce al lenguaje de la máquina objetivo usando el patrón y c) el código máquina se transfiere del patrón al objetivo.

El paso a) requiere un patrón con buena gestión de ficheros y facilidades de edición.

Se usa un *sistema operativo* para almacenar y gestionar los ficheros y un *editor* para entrar y modificar el texto.

El paso b) requiere unos programas llamados ensambladores-cruzados y compiladores-cruzados. Un *ensamblador-cruzado* es un programa que se ejecuta en la máquina patrón pero traduce el código fuente en lenguaje ensamblador al lenguaje máquina de la máquina objetivo. Un *compilador-cruzado* es un programa que se ejecuta en la máquina patrón pero traduce el código fuente en un lenguaje de *alto nivel* a código máquina de la máquina objetivo. A medida que se sofistican las máquinas la diferencia entre ambos programas es cada vez menor. Por ejemplo, el nuevo Intel iAPX 432 está diseñado para programarse en un lenguaje de muy alto nivel, el Ada.

El paso c) requiere un método de comunicación entre el patrón y el objetivo. Un método común es una línea de datos en serie, quizás usando el chip Controlador Programable Serie de Interfaz 8251 de Intel en los dos extremos de la línea. Tanto en la máquina objetivo como en la patrón se necesitan pequeños programas que controlen las transferencias.

Dada la sofisticación y grado de difusión alcanzados por los sistemas basados en el 8080/8085, actualmente se sigue el segundo enfoque (patrón-objetivo) en el desarrollo de software para el 8086/8088. Antiguamente, cuando se utilizaba el segundo enfoque se usaba como patrón normalmente una computadora más grande y potente que el objetivo, incluso una minicomputadora o una maxicomputadora. Esto se hace aún, pero ahora se ve más práctico enfocarlo desde el otro lado. Los sistemas basados en el 8080/8085/Z80 tienen sistemas operativos muy potentes (como el CP/M de Digital Research), que pueden mantener gran número de grades ficheros, almacenados en discos flexibles. Además existen editores potentes de fácil uso, que trabajan bajo esos sistemas operativos. Es más fácil utilizar estos editores que hacerlo a base de «lápiz y papel», y en muchos casos son más fáciles de usar que los editores de las grandes computadoras.

Otro factor que hace hoy en día mucho más viable este segundo enfoque (patrón-objetivo), es la gran cantidad de software en forma de código fuente desarrollado y funcionando en máquines 8 bits. La mayoría puede trasladarse, o bien manualmente, o de una forma casi automática (usando un programa de conversión) a una de las nuevas máquinas de 16 bits. Sin embargo, hay razones importantes para que este enfoque no sea siempre el más adecuado. Una es que el código traducido línea a línea es realmente menos eficiente que el original, tanto en ocupación de memoria como en velocidad. Otra razón es que las nuevas máquinas de 16 bits son más potentes que las de 8 bits por lo que deben tomarse nuevos caminos en la resolución de los problemas. El primer tema con el que empezar es el sistema

operativo. La mayoría de sistemas operativos para 8 bits realizan una única tarea en un momento dado. El operador debe esperar en cada paso. Por otro lado, las nuevas máquinas de 16 bits soportan varias tareas a la vez y los nuevos sistemas operativos deben usar esta gran ventaja. Por ejemplo, el XENIX, que está siendo desarrollado por Microsoft a partir del sistema operativo UNIX, originalmente desarrollado en Bell Labs, permitirá que un trabajo genere otros trabajos. Todos los trabajos se ejecutarán concurrentemente en este sistema y podrán generar a su vez más trabajos, resultando que gran cantidad de tareas se realizarán concurrentemente.

LOS PRIMEROS CUATRO AÑOS

El primer fabricante que desarrolló software y hardware para los procesadores 8086, 8088, 8087 y 8089 fue la propia Intel. Reconociendo la necesidad de dar soporte a estos chips, Intel invirtió gran cantidad de dinero en un gran y moderno edificio en Santa Clara, California, dedicado al diseño, fabricación y venta de sus *sistemas de desarrollo*. Un sistema de desarrollo es una computadora autosuficiente con el software y hardware necesario para desarrollar software de microprocesadores.

Los sistemas de desarrollo son factores clave para asegurar las ventas de una empresa fabricante de chips. La inmensa mayoría de ventas son a otras empresas, las cuales usan estos chips en aparatos electrónicos, diseñados, fabricados y comercializados por ellas mismas. El chip provee la materia prima para el trabajo a nivel de dirección o sistema. A las empresas que usan los productos de otros fabricantes para desarrollar sus propios productos se les llama fabricantes de equipo original, o de forma abreviada OEM (Original Equipment Manufacturers). El disminuir el tiempo de desarrollo de software y hardware resulta esencial para las OEM, ya que el mercado de estos productos es muy competitivo. Estas compañías compran a los fabricantes de chips que les dan el mejor soporte. Necesitan soporte ya que los seis meses que puede requerir el que ellos desarrollen las herramientas apropiadas, les puede significar unas pérdidas de millones de dólares en ventas. Quieren ser los primeros fabricantes en el mercado, con lo cual pueden asegurarse la mayoría de ventas en dos áreas importantes: 1) a corto plazo, ya que al principio la demanda es mucho mayor que la oferta, y 2) a largo plazo, ya que el primer producto marca, a menudo, los estándares.

Intel utiliza, básicamente, el enfoque patrón-objetivo. Ha desarrollado y realizado una serie completa de software que se ejecuta en su *Intellec Microcomputer Development System,* basado en el 8085. Los programas incluyen: dos ensambladores, el ASM86 (para el 8086/8088 y el 8087) y el ASM89 (para el 8089); tres lenguajes de alto nivel, PL/M-86, FORTRAN y Pascal; y varios programas de ayuda, LINK86, LOC86, OH86 y LIB86. Hay asimismo un programa traductor, el CON V86 que transforma códigos fuentes en ensamblador del 8080/8085 a código fuente ensamblador del 8086/8088. El patrón es su *Intellec Microcomputer Development System* y el objetivo es su *iSBC 86/12 Single*

Board Computer.Ha sido la propia Intel la que ha realizado la transición gradual de la CPU 8086 de 16 bits, al sistema de desarrollo. El *Series III Intellec Development Systems* tiene en este momento dos versiones: una que contiene la antigua CPU 8085 y otra que alberga a la nueva CPU 8086 de 16 bits, que puede formar pareja con el NDP 8087. También hay dos versiones del software de desarrollo de Intel, una por càda lado, y hay órdenes de paso de una a otra. Gradualmente, cierto software como ensambladores, editores de enlace, y lenguajes de alto nivel han elegido su vía al lado del 8086.

Otras compañías ofrecen sistemas basados en el 8086/8088, o software para estos sistemas, o ambas cosas. Las tablas 8.1,.8.2 y 8.3 muestran muchos de los productos disponibles en el momento de escribir este libro.

Fabricante	Producto
Action Computer Enterprises	Placa CPU 8086
Apparat, Inc.	Productos generales para el IBM
A.S.T Research	Tarjetas de memoria y comunicaciones para IBM
Cal-Tech Computer Services	Placa 8088 para el Apple
Chrislin Industries, Inc.	Memoria para IBM
Godbout	Placa procesadora dual 8085/8088
Godbout	Placa de procesamiento paralelo 8086/8087
IBM	Computadora personal IBM (basado en el 8088)
Intel	Sistema de desarrollo Intellec Serie III
Intel	Placa computadora simple iSBC/12 (CPU 8086)
Intel	Módulo de diseño de sistemas SDK-86
Lomas Data Produts	Placa CPU LDP88 y sistema completo
Lomas Data Products	Placa CPU 8086/8087/8089
Metamorphic Systems, Inc.	Placa 8088 para el Apple
Seattle Computer Products	Placa CPU 8086 y sistema completo
TecMar	Placa CPU 8086 y sistema completo
TecMar	Caja de expansión para el IBM
TecMar	Productos generales y específicos para el IBM

Tabla 8.1: Algunos de los equipos basados en el 8086/8088.

Godbout y Microsoft fueron de las primeras compañías en introducirse en el mercado, tanto personal como profesional, con productos que basados en la CPU 8086/8088. La placa Procesador-Dual 8085/8088 de Godbout provee una forma barata de tener tanto el objetivo como el patrón en el mismo sistema. El ensamblador cruzado 8086 de Microsoft provee el software para usar dicho procesador-dual. Si se tiene un bus principal S-100, se puede reemplazar la CPU 8080, 8085 o Z80 por el procesador-

Fabricante	Producto	Tipo
Digital Research	CP/M-86	Sistema operativo y utilidades
Digital Research	MP/M-86	Sistema operativo
Intel	IRMX88	Sistema operativo
Intel	IRMX86	Sistema operativo
Industrial Programming	MTOS-86	Sistema operativo
Hemenway	SP/8086	Sistema operativo y utilidades
Hemenway	MSP/8086	Sistema operativo y utilidades
Phase One Systems	OASIS-8086	Sistema operativo
Systems & Software	REX-80	Sistema operativo
Microsoft	XENIX-8086	Sistema operativo
Seattle Computer Products	86-DOS	Sistema operativo y utilidades
IBM	DOS	Sistema operativo
Intel	ASM86	Ensamblador 8086/8088/8087
Intel	LINK86, LOC86	Utilidades 8086
Intel	ASM89	Ensamblador 8089
Microsoft	XMACRO-86	Ensamblador cruzado 8086/8088
Microsoft	MACRO-86	Ensamblador 8086/8088
Intel	CONV86	Programa de conversión
Sorcim	TRANS 86	Programa de conversión
Intel	PL/M 86	Compilador PL/M
Digital Research	CBASIC-86	Intérprete BASIC
Microsoft	BASIC 86	Intérprete BASIC
Microsoft	BASIC 86	Compilador BASIC
Intel	FORTRAN 86	Compilador FORTRAN
Microsoft	FORTRAN-86	Compilador FORTRAN
Microsoft	COBOL-86	Compilador COBOL
Cybernetics, Inc.	RM/COBOL	Compilador COBOL
Microsoft	Pascal-86	Compilador Pascal
Sorcim	Pascal/M86	Compilador Pascal
Intel	Pascal 86	Compilador Pascal
MT Micro Systems	Pascal/MT+	Compilador Pascal
SofTech Microsystems, Inc.	UCSD Pascal	Pascal y sistema operativo
Intermetrics	PasPort 8086	Compilador cruzado Pascal PDP-11
Computer Innovations, Inc.	C86	Compilador C
Supersoft	C (8086 vers)	Compilador C
Information Unlimited Software	Easy-Writer	Editor de textos
CompuView	VEDIT	Editor de textos

Tabla 8.2: Algunos productos software basados en el 8086/8088.

dual Godbout, que contiene una CPU 8085 y una 8088. Bajo control software se puede pasar de una a otra CPU. Se puede reemplazar el sistema usando la CPU 8085 y se desarrolla el software del 8088 almacenándolo en los propios discos flexibles. Se carga luego en memoria y ¡se pasa el sistema a 8088! El ensamblador cruzado XMACRO-86 de Microsoft puede ser de gran ayuda durante este proceso, pero también se puede desarrollar el ensamblador a mano si se es, al igual que nosotros al principio, algo masoquista. El ensamblador-cruzado *se ejecuta* bajo el sistema operativo CP/M, y produce código para el 8086/8088. Nosotros usamos este ensamblador-cruzado en con-

Action Computer Enterprises, Inc.
55 West Del Mar Boulevard
Pasadena, CA 91105

Apparat, Inc.
4401 So. Tamarac Parkway
Denver, CO 80237

A.S.T. Research Inc.
17925 B Skypark Circle
Irvine, CA 92714

Cal-Tech Computer Services Inc.
4112 Napier St.
San Diego, CA 92110

Computer Innovations, Inc.
75 Pine St.
Lincroft, NJ 07738

CompuView
1955 Pauline Blvd.
Suite 200
Ann Arbor, MI 48103

Crislin Industries, Inc.
31352 Via Colinas
Westlake Village, CA 91362

Cybernetics, Inc.
8041 Newman Ave.
Suite 206
Huntington Beach, CA 92647

Digital Research
801 Lighthouse Rd.
Pacific Grove, CA 93950

Godbout Electronics
Box 2355
Oakland Airport
Oakland, CA 94614

IBM Corporation
Information System Division
Entry Systems Business
Boca Raton, FL 33432

Industrial Programming, Inc.
100 Jerico Quadrangle
Jericho, NY 11753

Information Unlimited Software, Inc.
281 Arlington Ave.
Kensington, CA 94707

Intel Corporation
3065 Bowers Ave.
San Clara, CA 95051

Intermetrics
733 Concord Ave.
Cambridge, MA 02138

Hemenway Associates
101 Tremont St.
Boston, MA 02108

Lomas Data Products
11 Cross Street
Westborough, MA 01581

Metamorphic Systems, Inc.
P.O. Box 1541
Boulder, CO 80306

Microsoft
10800 NE 8th St.
Bellevue, WA 98004

MT Micro Systems
1562 Kings Cross Drive
Cardiff, CA 92007

Phase One Systems, Inc.
770 Edgewater Dr.
Suite 710
Oakland, CA 94621

Seattle Computer Products, Inc.
1114 Industry Drive
Seattle, WA 98188

SofTech Microsystems, Inc.
9494 Black Mountain Road
San Diego, CA 92126

Sorcim Corporation
405 Aldo Ave.
Santa Clara, CA 95050

Supersoft Associates
P.O. Box 1628
Champaign, IL 61820

Systems & Software, Inc.
2801 Finley Rd.
Donnors Grove, IL 60515

TecMar Inc.
23600 Mercantile Rd.
Cleveland, OH 44122

Tabla 8.3: Algunos fabricantes de hardware y software.

junción con el Procesador-Dual Godbout para desarrollar y probar los ejemplos del último capítulo.

Digital Research posee ahora una versión de CP/M (Control Program for Microcomputers: Programa de Control para Microcomputadoras) para el 8086/8088, llamado CP/M-86. (Véase Murtha, Stephen M.; Waite Mitchell *CP/M Primer*. Indianápolis, Indiana. Howard W. Sams & Co., Inc.). El CP/M-86 es muy similar al CP/M basado en el 8080/8085/Z80 (ahora llamado CP/M-80). Al igual que su predecesor CP/M-80, el nuevo CP/M-86 tiene un ensamblador (ASM), un editor (ED), un programa de transferencia de ficheros (PIP, abreviatura de Peripheral Interchange Program), y una herramienta de depuración de programas (DDT, abreviatura de Dynamic Debugging Tool). Hay también una versión especial de ensamblador que actúa como ensamblador-cruzado desde el sistema 8080/8085/Z-80 al nuevo sistema 8086/8088. Resulta una buena ayuda si se utiliza una máquina de 8 bits para inicializar un sistema de 16 bits. Los discos producidos en un sistema CP/M-80 deben ser legibles por un sistema CP/M-86 (con la misma unidad de disco). Es interesante notar que el ensamblador del CP/M es muy similar al ensamblador de Intel, dependiendo más del tipo de los operandos que de los nemotécnicos especiales (como añadir la letra I) para determinar la diferencia entre datos inmediatos y las direcciones de desplazamiento para variables.

Todavía queda un problema para que este sistema operativo trabaje en el sistema: falta de compatibilidad entre las diferentes unidades de disco. El CP/M-86 estuvo disponible al principio sólo para discos flexibles de densidad simple de 8 pulgadas. Por ello, si se tenía un sistema de 5 1/4 pulgadas (que es el otro tamaño estándar de disco flexible) no se podía acoplar el CP/M en la unidad de disco. Como regla, casi todos los discos flexibles de densidad simple de 8 pulgadas pueden leerse en cualquier unidad de 8 pulgadas en un sistema microprocesador. La situación es algo diferente en el mundo de las 5 1/4 pulgadas. Hay discos flexibles de 16 sectores, de 10 sectores y un (-soft) sector. La mayoría de unidades de 5 1/4 pulgadas no pueden leer otros tipos que no sea el correcto. Realmente, esto sólo es el principio del problema, ya que se puede tener una copia de cualquier disco flexible en formato de 8 pulgadas. Hay empresas de servicios que hacen estas copias por dinero, aunque siempre se puede tener a un amigo que conozca a otro amigo que tenga un sistema con ambos tipos de unidad de discos flexibles y el software para realizar la copia. El problema real es estar seguro que el CP/M sepa cómo tratar el disco flexible. Hay una sección especial del CP/M llamada BIOS (Basic I/O System: Sistema Básico de E/S) que debe configurarse para cada sistema de computadora diferente. El BIOS tiene dos partes semidistintas: las rutinas de interfaz consola/impresora y las rutinas de interfaz disco-unidad de disco. Escribir uno mismo las rutinas de consola/impresora es algo bastante sencillo. Si se ha leído todos los capítulos de este libro, probablemente esté capacitado (con la ayuda de los manuales de

CP/M) para escribir su propia versión. Por otro lado, las rutinas de disco son, a menudo, complicadas y pueden escribirlas únicamente personas muy expertas. Esta dificultad varía mucho según lo inteligente que sea el controlador de la unidad de disco (ver el capítulo de chips de soporte). La mayoría de personas esperan a que Lifeboat Associates u otra compañía saque la versión del CP/M-86 de su sistema, cosa que empieza a ocurrir ya.

Una vez que CP/M-86 y otros sistemas operativos de disco estén disponibles en los diferentes formatos, existiendo interfaz entre los diferentes tipos de unidad de disco ya no tiene por qué existir ninguna laguna con otros tipos de software. Existen ya varias compañías que ofrecen versiones de Pascal (ver The Pascal Primer. Fox, David; Wait, Mitchell; Indianápolis, Indiana: Howard W. Sams & Co., Inc.) y de otros lenguajes como el C y BASIC. La mayoría de estas compañías ya tienen un lenguaje ensamblador para el correspondiente programa 8080/8085/Z-80. No será difícil convertirlos en programas para el 8086/8088. Sólo hace falta una cierta cantidad de tiempo, así que mejor espere.

Debe hacerse notar que el software de más alto nivel es prácticamente independiente de la máquina, o al menos, de la CPU particular en la que se vaya a ejecutar. Por supuesto, determinados equipos periféricos como pantallas video-gráficas tienden a hacer que el software de más alto nivel sea algo dependiente de la máquina, pero incluso aquí hay ciertos estándares. Como resumen, si se tiene un programa BASIC, FORTRAN, C, Pascal o incluso FORTH que funciona en un sistema con un bus S-100, debe seguir funcionando sin necesidad de cambios al pasar al procesador 8086 u 8088 y obtener el nuevo sistema operativo y compilador o intérprete correspondiente.

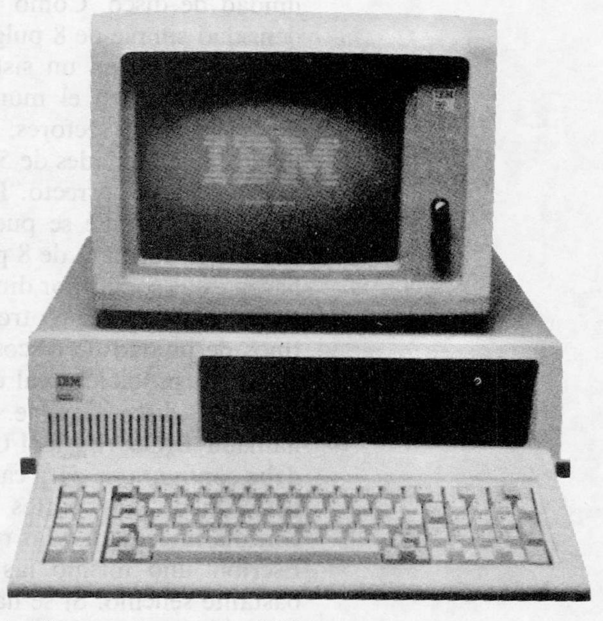


Figura 8.1a: Computadora personal IBM.

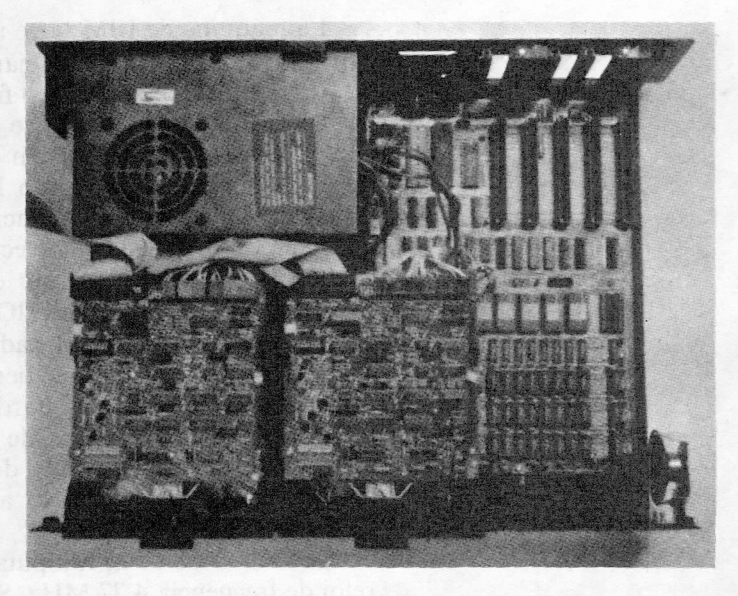


Figura 8.1b: Vista interior de la computadora personal IBM.

EL IBM «ACORN»

Uno de los desarrollos más notables para la familia de chips procesadores Intel 8086 es la elección de la CPU 8088 por parte de IBM cuando en 1981 entró en el área de las computadoras personales. Esta computadora se desarrolló bajo un doble proyecto con el nombre «ACORN» (Proyecto «BELLOTA») pero se está vendiendo bajo un nombre menos imaginativo, aunque quizás más correcto: «Computadora Personal IBM». La computadora, con un precio que varía desde unos 1.260 dólares hasta aproximadamente 3.830 dólares (con 48 K y una unidad de discos flexibles cuesta 2.235 dólares en Computerland), entra en competencia directa con los ofrecidos por Apple y Radio Shack. La unidad básica utiliza una casete normal para almacenamiento y una televisión como pantalla. La versión más cara incluye hasta dos unidades de discos flexibles (cerca de 160 K octetos cada uno) y monitores de color. Por supuesto, recuérdese que el software es un coste adicional pero necesario. De hecho se pueden gastar casi 3.000 dólares si se compra el BASIC avanzado de Microsoft (40 dólares), el Compilador Pascal de Microsoft (300 dólares), el Easy Writer de Information Unlimited (175 dólares), el General Ledger de Peachtree Software (595 dólares), el Accounts Recervable de Peachtree Software (595 dólares), el Account Payable de Peachtree Software (595 dólares), el soporte de comunicaciones asíncronas (40 dólares), el Adventure de Microsoft (30 dólares) y el Advance Diagnostics Package (155 dólares).

IBM tiene un interfaz para casetes que posibilita que éstos puedan usarse como medio de almacenamiento si no se compra el sistema de discos flexibles. Sin embargo, se debe obtener el casete aparte.

La máquina de IBM tiene un bus principal con un sub-bus de datos de 8 bits (para emparejarse con el bus de datos de 8 bits de la CPU del 8088). Hay cinco franjas de expansión para memoria extra; E/S, u otra cosa que se desee. Esto no es el bus S-100. De hecho, sólo hay 62 señales en el bus de IBM. La mayoría de las señales del 8088, tales como las 20 líneas de direcciones, las 8 líneas de datos, la de habilitación de latch de direcciones (ALE), la de dirección activa [del Selector de Bus 8089 (AEN)], lectura en memoria (MEMR), escritura en memoria (MEMW), lectura de E/S (IOR), escritura de E/S (IOW), y reloj (CLOCK) aparecen en el bus de forma demultiplexada o «separada», y además hay un juego de señales nuevas que tienen que ver principalmente con las facilidades de interrupción hardware del IBM. IBM está publicando los estándares precisos de su bus y alienta a los pequeños fabricantes de placas a que desarrollen sus propias placas que cubran dichas franjas. Ya hay anuncios de productos complementarios para el IBM.

La CPU 8088 de la computadora IBM funciona con pulsos de reloj de frecuencia 4,77 MHz. Sin embargo, la memoria funciona a 2,44 MHz. Ello está en línea con la sugerencia de Motorola de que su MC68000 funcione a una velocidad más rápida que el resto del sistema. Dado que el coste de la memoria aumenta con su velocidad, y dado que hay varias instrucciones (en particular multiplicar y dividir) en la CPU 8088 que necesitan un gran número de ciclos de reloj para proceso interno, el coste de la memoria se mantiene bajo mientras se mantiene un alza en el rendimiento. Por ello, IBM piensa ofrecer placas de expansión de memoria a precios bastante razonables (16 K octetos por 90 dólares, 23 K octetos por 395 dólares y 64 K octetos por 540 dólares). El tiempo de acceso a memoria se supone que está alrededor de 250 ns, lo cual corresponde a un reloj «virtual» de velocidad 4 MHz, y a la velocidad estándar para un Z-80. La memoria está organizada en palabras de 9 bits, con un bit extra para detectar errores (ver la explicación de detección ycorrección de errores en el capítulo 4). Hasta 256 K octetos de RAM de usuario pueden añadirse al sistema, considerablemente mayor que los 64 K octetos disponibles normalmente para los antiguos procesadores de 8 bits con su direccionamiento de 16 bits. Esto es menor que una capacidad de direccionamiento de 1 megaocteto de la CPU 8088, pero el sistema IBM usa algunos de los rangos de dirección «perdidos» para funciones del sistema como los 40 K octetos ROM para intensificar el BASIC de Microsoft y los 16 K octetos de ROM video pantalla.

La computadora está diseñada para que su manejo sea sencillo. En particular, el teclado puede ponerse en las rodillas o en una mesa, con varios ángulos de inclinación. Pueden usarse letras mayúsculas y minúsculas, por lo que puede utilizarse como procesador de textos. Tiene además diez teclas de funciones especiales para su uso como procesador de textos. Cada tecla envía realmente dos códigos de función. Envía un código al presionar la tecla y otro al levantarse. Internamente la computa-

dora utiliza el código ASCII, lo cual es algo bastante nuevo en IBM (IBM inventó su propio código de caracteres: el EBCDIC).

Quizás la característica más interesante de la máquina IBM, respecto al uso de la CPU 8088, es su capacidad de gráficos de colores (disponible como extra con un adaptador especial). Tiene una RAM de 16 K octetos dedicada a la pantalla de video. Hay dos formatos de pantalla para gráficos: uno de resolución media de 320 pixels (puntos) horizontales por 200 verticales y uno de alta resolución de 640 pixels horizontales por 200 verticales. El formato de alta resolución sólo permite 1 bit por pixel mientras que el de media resolución admite 2 bits por pixel. Hay una forma de test en la cual cada carácter puede tener uno de los ocho colores intensos, uno de los ocho colores no intensos, así como estar parpadeando o no estarlo. Esto que parece imposible, se realiza almacenando 2 octetos para cada carácter de la pantalla. Un octeto contiene el código ASCII del carácter y el otro contiene la información de cómo representarlo.

Aunque las pantallas del modelo básico pueden ser unos televisores normales, se puede disponer de monitores de color con la opción de gráficos de colores. Dado que usa una señal estándar de televisión, hay un problema en incrementar la resolución vertical, mucho mayor de lo que pueda parecer. Por ejemplo, si se dobla la resolución vertical a 400 pixels, entonces las figuras empezarían a parpadear. Esto es debido a que las señales de televisión «pintan» una línea sí y otra no en cada imagen. Con lo cual se necesitan dos imágenes para mostrar la figura completa, estando la segunda imagen entrelazada con las líneas de la primera. Esto ocurre a una velocidad de 60 imágenes por segundo (para emparejar con los 60 ciclos de la corriente eléctrica). Por ello, cada pixel (punto) se «repinta» cada 1/30 segundos, con puntos verticalmente adyacentes apareciendo y desapareciendo de manera alterna. En una figura normal de televisión no es problema, pero cuando hay un gran contraste entre los pixels adyacentes, como en una figura generada por computadora, el parpadeo empieza a notarse. La solución es repetir la figura completa en cada imagen, y de esta forma la segunda imagen refuerza a la primera en vez de ir en contra de ella.

Acoplando un monitor de video opcional fosforescente verde y un adaptador distinto, el sistema trabaja como un procesador de textos excelente. El monitor representará 25 líneas de 80 caracteres. Características tales como subrayado, caracteres con alta intensidad, caracteres con parpadeo y caracteres en negativo están disponibles bajo control del software. Este monitor especial es realmente necesario dado el problema del parpadeo y la poca capacidad de las televisiones normales para representar 80 caracteres horizontales. Para eliminar el parpadeo, el monitor está equipado especialmente para mantener la luz más tiempo, cada vez que se activa. Se usa una fosforescencia *muy persistente*. Este tipo de fosforescencia producen normalmente una imagen verde.

IBM se está introduciendo directamente en el mercado de software de computadoras personales para este nuevo computa-

dor. Ha encargado a Microsoft el realizar tres versiones de BASIC: BASIC para casete, BASIC para disco y BASIC avanzado. Las tres tienen características muy interesantes. Quizás la más fascinante es la habilidad para transmitir gráficos de color en forma de cadena. Por ejemplo, la cadena:

A$=«r10;d30;120;u10»

definiría una caja (derecha 10 unidades, abajo 30 unidades, izquierda 10 unidades y arriba 30 unidades). Esta cadena se puede combinar con otras cadenas para formar esquemas más elaborados. La orden DRAW del BASIC hace que se dibujen las cadenas: por ejemplo,

DRAW A$

haría que la caja se dibuje sobre la pantalla.

IBM ofrece un sistema operativo especial, similar al CP/M, llamado Computadora Personal IBM-DOS. Este sistema fue desarrollado por Microsoft para IBM, dada la dificultad de que el CP/M-86 pasara el control de calidad de IBM. El DOS de IBM tiene las mismas llamadas de sistema que el CP/M-86 (llamadas de los programas de aplicaciones al sistema operativo). Sin embargo, hay mejoras sustanciales respecto al CP/M. Por ejemplo, hay un sistema de edición incorporado para entrar órdenes. Cada orden crea una plantilla para la orden siguiente, por lo que si se comete un error que sólo afecta a unos pocos tecleos, no se tendrá que volver a teclear toda la orden. Incluso si no se cometen errores (¡¿!), esta característica será muy práctica para entrar secuencias de órdenes similares.

En su marketing inicial (octubre, 1981), el DOS de Microsoft soportaba un juego completo de software orientado a aplicaciones, incluyendo un compilador Pascal de Microsoft, Visical de Personal Software, Easywriter (procesador de textos de Information Unlimited, y un programa de comunicaciones asíncronas escrito en BASIC que IBM prometió expandir para que la computadora pudiera simular un terminal IBM-3270. Adicionalmente hay un paquete empresarial que incluye **contabilidad** general, cuentas a pagar y cuentas a cobrar, todo de Peachtree Software. IBM ofrece además otros dos sistemas operativos: CP/M-86 de Digital Research y UCSP p-System, el cual incluye el famoso UCSD Pascal.

IBM creó una división, únicamente para manipular esta computadora personal y su software. Está animando el desarrollo de nuevo software en un ambiente de mercado abierto dentro del espíritu del mercado actual de microcomputadoras.

CONCLUSIONES

Hemos explorado los usos corrientes de la familia de chips procesadores 8086. Muchas compañías, además de Intel, han dado un fuerte apoyo y soporte para las CPU 8086 y 8088,

proveyendo los sistemas básicos y el software para estos procesadores.

Estas compañías incluyen a: Godbout (Procesador Dual 8085/8088), Seattle Computer (Procesador 8086), Digital Research (CP/M-86), Microsoft (XMACRO-86, BASIC-86 y eventualmente XENIX) e IBM (el Computador Personal). En particular, la elección de la CPU 8088 por parte de IBM para su inicio en la competición de las computadoras personales promete la apertura de una nueva era en las computadoras personales y profesionales. El enfoque de IBM debe llegar a producir gran cantidad de software de aplicación e interesantes periféricos hardware.

Apéndice A

El iAPX 186 y el iAPX 286

En estos momentos Intel está desarrollando ciertas mejoras y extensiones del iAPX 86 y del iAPX 286. El primero es el iAPX 186, el cual es realmente una versión «más rápida» del iAPX 86, y el segundo es el iAPX 286, que extiende el hardware del chip y el juego de instrucciones del iAPX 86 en las áreas de gestión de memoria y memoria virtual.

EL iAPX 186

Empecemos con el iAPX 186. El iAPX 86 fue el primero de la nueva generación de máquinas de 16 bits, saliendo al mercado antes de que Motorola y Zilog entraran en este campo. En comparación con estos procesadores más nuevos, el iAPX 86 es algo menos eficiente en cierto tipo de instrucciones como las de tratamiento de cadenas, multiplicación, división, desplazamiento y rotación. Con el anuncio del iAPX 186 como su producto de cara al futuro, Intel parece haber reconocido estas limitaciones, y ha prometido una nueva versión del 8086 que ejecutará estas instrucciones a unas velocidades más competitivas con los otros dos procesadores. Por ejemplo, la multiplicación y división en el 186 es unas tres o cuatro veces más rápida que en el 8086, y los movimientos de cadena repetitivos se ejecutan ahora tan rápido como los buses lo pueden transportar, lo que hace que dichos movimientos sean más bien unas transferencias DMA.

Como siempre, la nueva oferta tiene algunas nuevas y agradables sorpresas, así como el retorno de algunas características antiguas que eran deseadas. Para el iAPX 186, las nuevas características incluyen un control de interrupciones multinivel, dos canales DMA de alta velocidad y tres relojes programables de 16 bits. En otras palabras, el 186 integra en el procesador muchas de las características de la familia de chis soport iAPX 86, reemplazando toda la placa circuito por pequeñas fracciones de silicio de una pulgada cuadrada. Para el 186, las características antiguas incluyen el mismo núcleo de instrucciones del 8086 (con pocas adiciones) más un generador de pulsos de reloj integrado tal como lo tenía el 8085.

El iAPX 186 se ha pensado para un amplio rango de aplicaciones, incluyendo terminales inteligentes, computadores para pequeñas empresas, sistemas de entrada de datos, sistemas de control de procesos y subsistemas de E/S para computadoras grandes, como el sistema iAPX 432. Esto enlaza con el tipo de aplicaciones para las cuales ya se utiliza el 8086. El 186 facilita y baja el coste de estas aplicaciones al reducir el número de chips

necesarios en estos sistemas, lo cual a su vez, reduce el tamaño físico y la complejidad de este tipo de sistemas.

Las instrucciones adicionales del iAPX 186 incluyen dos instrucciones (ENTER y LEAVE) para llamadas a subrutinas en lenguajes estructurados de alto nivel como el Pascal; dos instrucciones para introducir datos inmediatos (tanto 16 bits como 8 bits son ampliados automáticamente a 16 bits) en la pila del sistema, dos instrucciones para multiplicar datos inmediatos; una instrucción de verificación de fronteras de tablas (igual que el Motorola 68000); e instrucciones de desplazamiento y rotación en las cuales la cuenta del desplazamiento se especifica por datos inmediatos. También se incluyen las instrucciones PUSH ALL y POP ALL, muy útiles cuando se desea escribir subrutinas que hacen sus propias tareas sin interferir con nada más. Por supuesto, las instrucciones PUSH ALL y POP ALL también son muy útiles al gestionar interrupciones en un sistema multitarea. Adicionalmente a las anteriores instrucciones, hay instrucciones para mover bloques en las cuales la fuente o el destino puede estar en el espacio de E/S.

El iAPX 186 tiene algunas características más, aparte de las que se han descrito, pero lo que aquí se ha presentado debe dar una buena idea de lo que puede hacer el iAPX 186, que es todo lo que hacen el 8086 y más (y alrededor de un 30 por 100 más rápido), ¡con bastantes chips de soporte menos!

EL iAPX 286

El iAPX 286 ha añadido un nuevo nivel de satisfacción a la arquitectura básica del 8086, incluyendo una gestión de memoria como la extensión natural de las capacidades de direccionamiento del procesador. El 286 tiene elaboradas facilidades incorporadas de protección de datos, colocando su seguridad a mitad de camino entre el 8086 (ninguna) y el 432 (desarrollado en el siguiente apéndice). Otras características del 286 incluyen todas las características del juego de instrucciones del nuevo 186, así como la extensión del espacio de memoria direccionable a 16 megaoctetos (usa 24 bits para direccionar). ¡Se comporta la memoria virtual con un espacio direccionable de alrededor de un gigaocteto!

Al igual que el 432, el 286 revisa cada acceso a instrucciones o datos para comprobar si puede haber una violación de los *derechos de acceso*. Al revés que el 432, el 286 está diseñado para usar un sistema operativo convencional con varios niveles de privilegio. En este tipo de sistemas operativos hay un *núcleo* que, como su nombre indica, es la parte más interna del sistema operativo. El núcleo tiene el máximo privilegio y los programas de aplicación el mínimo. Es interesante el que la estructura del sistema operativo del 286 sea llamada «sujeto» por Intel. Esto contrasta con el nombre «objeto», el cual se usa para describir las estructuras fundamentales de operación del iAPX 432. El 286 permite cuatro niveles de privilegio. La protección de datos en este tipo de sistemas se lleva a cabo teniendo instrucciones, datos y segmentos códigos privilegiados así como derechos de acceso para cada segmento del sistema.

Para un usuario normal, los registros de segmentación (segmento código, segmento datos, segmento extra y segmento pila) parecen tener los 16 bits usuales. Sin embargo, estos 16 bits realmente no apuntan directamente a memoria como lo hacen en el 8086. En su lugar, apuntan a tablas especiales, llamadas tablas descriptoras, algunas de las cuales tienen que ver con los usuarios y otras con el sistema operativo. Actualmente a estos 16 bits, cada registro de segmentación del 286 mantiene otros 57 bits transparentes para el usuario. Ocho de estos bits los utiliza el hardware para mantener los derechos de acceso (sólo lectura, sólo ejecución y otros), otros bits mantienen la dirección real (24 bits en este momento) del principio del segmento y otros mantienen la longitud permitida del segmento (hasta 64K octetos, especificados por 16 bits para acceder a 16 megaoctetos). Por ello, al usuario nunca se le informa de si está trabajando en memoria y siempre se mantiene dentro de ciertas fronteras. Como característica adicional, nunca se permite que el usuario escriba en el segmento código. Ello previene que el usuario modifique un programa para realizar actos ilegales y potencialmente peligrosos. Hay también provisiones para prever que el usuario introduzca en el sistema un «caballo de Troya» que pueda proporcionarle un estado de alto privilegio.

El 286 tiene tres nuevos registros. Estos apuntan a las tablas descriptoras actualmente en uso. Estas tablas contienen la información sobre los objetos protegidos en el sistema. Cualquier cambio de privilegio o de segmento debe hacerse a través de dichas tablas. Adicionalmente hay varios indicadores nuevos (de 1 bit) en el hardware del 286.

El 286 tiene varias instrucciones nuevas además de las del 186. Todas estas instrucciones se refieren a la gestión de memoria a protección del sistema haciendo cosas tales como cargar y almacenar el contenido de los indicadores especiales y de los punteros, incluyendo las tablas descriptoras y sus punteros.

CONCLUSIONES

Es alentador el ver los anuncios de Intel sobre estos nuevos productos. Dada su compatibilidad con los actuales chips 8086/8088, los anuncios de estos nuevos productos significa que ya es provechoso invertir tiempo y esfuerzo en el 8086/8088. Por ello, tendencias y programas (código de la computadora) desarrollados para el 8086/8088 serán apropiados, con muy pocas modificaciones, para el hardware más potente del futuro.

Apéndice B

El microprocesador Intel iAPX 432 de 32 bits

Este apéndice da una visión general de uno de los desarrollos modernos más interesantes en el campo de la arquitectura de computadoras; la familia de microprocesadoras iAPX 432. El 432 no se limita a ser la respuesta de Intel a los microprocesadores «pseudo 32 bits» de otros fabricantes. Es, de hecho, el punto de partida de las tendencias actuales en arquitectura de microprocesadores debido por un lado a que su hardware funciona a un nivel más alto que los anteriores y por otro a que esta familia permite que trabajen cooperativamente pero independientemente muchas copias (¡hasta 256!) del mismo microprocesador, formando el núcleo de una computadora mucho mayor.

El iAPX 432 es el primer microprocesador que implementa algunas investigaciones teóricas recientes (algunas de ellas guardadas en secreto) sobre cómo aprovechar los avances en el área del diseño de hardware de computadoras para resolver algunos problemas reales y «agobiantes». Brevemente, estos problemas se centran en cómo ofrecer un servicio eficiente a gran número de usuarios a la vez que se asegura un alto grado de seguridad para los datos «delicados».

En este apéndice comenzamos con una discusión general de los avances en el hardware de la computadora. las nuevas aplicaciones para sistemas de computadoras, y las nuevas necesidades del diseño generadas por dichas aplicaciones. Estudiaremos después cómo funciona el iAPX 432 y varias maneras interesantes de satisfacer esas necesidades fundamentales.

NUEVOS AVANCES Y VIEJAS NECESIDADES

Avances

Aunque los avances en el diseño y fabricación de hardware no son problemas en sí, han abierto un amplio abanico de nuevas posibilidades. Hay mucho que aprender, y muchos problemas que vencer todavía; pero muchos conceptos y métodos hoy en día prácticos, eran impensables hace muy poco tiempo.

Los avances en el diseño y fabricación de chips para computadoras han permitido encapsular tremendas capacidades de cálculo en chips cada vez menores y más baratos. Hoy en día se pueden introducir más de 100.000 dispositivos electrónicos primitivos en un único chip más pequeño que la uña del dedo meñique. En consecuencia, se fabrican sistemas de computadoras sustancialmente más sofisticados a un coste cada vez menor. Estos circuitos consumen menos y disipan menos potencia que los antiguos, con lo que el coste del circuito en sí mismo y de los sistemas de

soporte que lo albergan, alimentan y enfrían es menor. Como estos sistemas de soporte suelen ser metálicos y tienen un cierto número de partes móviles, cualquier reducción de tamaño o capacidad redunda en el abaratamiento del sistema total.

Los mismos avances aparecen en la fabricación de memorias de gran capacidad a precios muy razonables. Lejos están los días en los que a cada bit de memoria se le debía pasar individualmente un arrollamiento de hilo conductor. Hoy en día las memorias se construyen con la misma tecnología que los modernos microprocesadores, permitiendo altas densidades, bajos consumos y bajos costes comparables.

Antiguamente los sistemas de computadoras se diseñaban con una potencia de proceso limitada y con las memorias de baja capacidad disponibles en aquel tiempo. Las estrategias actuales son totalmente distintas. Se necesitan en particular nuevos enfoques en el diseño de sistemas operativos, ya que la mayor capacidad de las memorias posibilita el que éstos puedan ser mucho más sofisticados. Veremos a lo largo del apéndice cómo el iAPX 432 ha incorporado en su hardware algunos de estos nuevos enfoques.

Necesidades

Al mismo tiempo que el hardware de semiconductores ha ido avanzando, ciertas áreas de aplicación como los bancos, las militares, las grandes empresas, la medicina o las aplicaciones aeroespaciales han comenzado a exigir sistemas de computadoras de alta-capacidad y alta-potencia-de-cálculo. Las necesidades básicas de diseño de sistemas de computadoras para estos usuarios puede resumirse en los siguientes puntos: 1) velocidad, 2) seguridad, 3) fiabilidad, 4) posibilidad de ampliaciones y 5) facilidad de programación. Estudiaremos en detalle cada uno de ellos.

Algunas de estas necesidades de cálculo se han estudiado durante mucho tiempo, llevando a sistemas limitados que, en vez de satisfacer los cinco puntos, eran aplicaciones altamente fundamentadas como ciertas instalaciones especiales gubernamentales. Ahora, con la continua disminución de los costes del hardware, de las memorias y la CPU, se hace posible el desarrollar sistemas de bajo coste que satisfagan mucho mejor dichos criterios. La razón precio/rendimiento está bajando hasta el punto que algunas aplicaciones comerciales impensables hace poco se están llevando a la práctica. El campo de batalla comercial es, sin duda, el que mueve más dinero y cuando algo impensable se hace posible, el dinero fluye en abundancia.

Veamos ahora cada uno de estos puntos con más detalle.

Velocidad

Típicamente, un sistema de computadora de tamaño medio o grande da servicio a varios usuarios que pueden estar utilizándolo a la vez, de una forma compartida. La computadora se diseña para que cada usuario tenga una «porción» de tiempo de proceso de la CPU, de manera que el usuario pueda ver el computador

como si en aquel momento estuviese trabajando sólo para él/ella. A la mayoría de los usuarios les gusta pensar que tienen toda la máquina para ellos y su trabajo no será retrasado por los programas de otros usuarios. Con esta utilización, el factor velocidad de la máquina se convierte en muy importante, y determina en gran manera la rapidez con que los usuarios podrán realizar sus trabajos. (Otro factor muy importante en la determinación de la velocidad en estas aplicaciones es la razón de transferencia de datos entre la computadora y los dispositivos de almacenamiento externo.) La velocidad de proceso tiene una importancia capital.

Para futuras aplicaciones a muchos usuarios les gustará sentir que están conectados a un sistema que tiene acceso a muchos recursos, incluyendo grandes bases de datos, una potencia de cálculo satisfactoria, o incluso a otros usuarios. El sistema de computadora se convertirá en un supermercado electrónico. En cualquier caso, los usuarios exigirán respuestas rápidas en contestación a sus demandas de servicio. Los usuarios, por supuesto, no esperarán que el servicio solicitado se realice instantáneamente, pero sí que la demanda sea aceptada por el sistema de una forma rápida y eficaz (segundos), y que los trabajos se procesen en un tiempo razonable (minutos). La aceptación de la demanda y su ejecución real son cosas distintas. La aceptación y tratamiento preliminar de las demandas es un proceso de tipo interactivo, mientras que la ejecución del servicio es más un proceso de tipo batch[1] (procesamiento por lotes). La solución para este tipo de sistemas multiacción implica el uso de multitareas (varios trabajos) con quizás varios procesadores distintos (varias CPU) en un único sistema. En este sistema, la potencia de cálculo no es tan importante como la organización del sistema. Con tantos trabajos realizándose a la vez, y dependiendo unos de otros, el computador se convierte en una especie de burócrata.

Si una parte del sistema queda «colgado», puede bloquear el avance de otras partes y disminuir la rapidez de proceso en muchas más. Esto sucede cuando los recursos como la CPU, memorias, o dispositivos periféricos están limitados, bien sea por falta de ellos, o por una mala estrategia de utilización. El iAPX 432 se diseñó para resolver estos problemas, proporcionando métodos que permiten dividir los trabajos mayores en tareas más pequeñas y tratables, ofreciendo una base de trabajo para unos esquemas eficientes de planificación de estas tareas y de las interacciones entre ellas, así como posibilitando la conexión fácil de varios procesadores y otros dispositivos al sistema computador. ¡De hecho, teóricamente, se pueden conectar hasta 256 unidades del 432 juntas para que trabajen cooperativamente en el mismo computador! Mientras estos procesadores no se interfie-

[1] El procesamiento *por lotes (batch)* es una estrategia del sistema operativo en el cual un usuario manda un trabajo completo y luego «desaparece» hasta que el trabajo se ha completado. El procesamiento *interactivo* es una estrategia en la cual el usuario y la computadora se comunican uno con otro durante la ejecución del trabajo.

ran, cuantos más procesadores conectemos, más rápidamente se servirán las demandas.

Seguridad

En grandes sistemas de computadoras en los que muchos usuarios trabajan a la vez, cada persona debe tener un cierto «perfil» de *derechos de acceso* a los datos del sistema. Un derecho de acceso es un permiso para leer o modificar una cierta porción de datos almacenados en un sistema computador. En general cada usuario tiene distintos derechos de acceso a los distintos bloques de datos del sistema. Por ejemplo, en un sistema de computadora de un banco, un cajero debería tener un conjunto de derechos de acceso completamente distintos que un vicepresidente, y, de hecho, los distintos vicepresidentes deberían tener diferentes derechos de acceso. Dicho de otra manera, para cada usuario hay un conjunto de datos que puede crear y modificar, otro conjunto de datos (normalmente mayor) que puede leer, y otro conjunto de datos al que no tiene ningún tipo de acceso. Todos estos conjuntos de datos para cada uno de los usuarios del sistema se solapan en formas complicadas, formando una matriz completa de derechos de acceso del sistema total. La violación de estos derechos puede causar pérdidas de miles de millones de dólares sea directamente, en el caso de bancos, o indirectamente, en el caso de la industria privada. Las violaciones pueden incluso «tambalear» la seguridad nacional, en el caso de sistemas del gobierno, o de ciertas industrias clave.

La seguridad no es un problema para los bancos ni para las instituciones de alto secreto («top-secret») gubernamentales. Pero los sistemas de computadoras hoy en día no se consideran totalmente seguros en otras muchas áreas. Por ejemplo, muchas universidades guardan información sobre las notas de sus estudiantes, o de tipo financiero, en los mismos sistemas de computadoras en los que programan los alumnos. La falta de seguridad en este caso puede ocasionar problemas legales.

En los sistemas actuales los usuarios tienen palabras de paso para proteger sus cuentas y ficheros de datos particulares. En los sistemas futuros las interacciones entre conjuntos de datos serán mucho más complejas, frecuentes y automatizadas, de forma que la gestión de las palabras de paso por seres humanos no será adecuada. En estos sistemas la computadora deberá asignar automáticamente palabras de paso para *cada* acceso a datos.

Veremos que el iAPX 432 utiliza un sistema operativo *basado en el objeto* con un esquema de direccionamiento *basado en la capacidad* que aumenta significativamente la seguridad. Con este sistema, a cada trabajo o tarea se le puede asignar sus propios derechos de acceso, y cualquier intento de violarlos genera inmediatamente una condición de error.

Fiabilidad

Además de las posibles violaciones de datos por parte de otros usuarios, los datos deberían estar protegidos contra destrucciones accidentales debidas a mal funcionamiento hardware o software.

Esto es, el procesador puede fallar o quedar colgado, o una cierta subrutina puede comenzar a realizar operaciones sobre un tipo de datos erróneo, por ejemplo realizando operaciones de coma flotante sobre sus propias instrucciones, o intentar ejecutar datos enteros como si fuesen instrucciones. Desde el momento que algunos de estos sistemas forman parte de equipos médicos, o de armas poderosas, fallos de este tipo pueden significar la muerte de seres humanos. ¡De todos nosotros en el caso de armas peligrosas!

Para la fiabilidad software, veremos cómo el sistema operativo basado-en-el-objeto del iAPX 432 lleva incorporadas medidas de protección para evitar estas tragedias.

Los datos en el iAPX 432 están *fuertemente clasificados,* con verificación automática (hardware) del tipo a cada acceso. Es decir, a cada unidad de información del sistema se le asigna un *tipo de datos*. Así, los enteros de 16 bits son un *tipo* de dato, los números en coma flotante de 80 bits son otro tipo de datos, y las instrucciones del procesador son otro tipo más. Incluso está previsto que el usuario pueda definir tipos de datos particulares. Cada acceso a un dato especifica el tipo de datos del que se trata, y cada tipo se guarda en un segmento distinto. Antes de acceder a un dato de un segmento, el iAPX 432 debe verificar si coinciden los tipos de datos del segmento y el especificado en la instrucción de acceso. Si el tipo de datos no es el esperado, el hardware produce automáticamente una condición de error. Para la fiabilidad hardware hay un modo de verificación especial en el cual las instrucciones se ejecutan repetidas en dos procesadores 432, y se comprueban los resultados.

Posibilidad de ampliación

Para aplicaciones pequeñas deben utilizarse sistemas computadores pequeños; para aplicaciones grandes se necesitan sistemas grandes. Es decir, la capacidad y, por tanto, el coste de un sistema debe ser proporcionado a las necesidades que se van a satisfacer. El problema es que han habido incompatibilidades entre sistemas con distintos niveles de actuación. Cada vez que se necesitaba un nuevo nivel de actuación, se requería un elaborado *proceso de conversión* que pasará los datos y programas del sistema antiguo al nuevo. Esto es contraproductivo, caro y lento. El iAPX 432 permite añadir nuevos procesadores al sistema existente ¡sin tener que cambiar el sistema operativo! Gracias a esto, se puede comenzar con un sistema sencillo uniprocesador y añadirle gradualmente procesadores (recuérdese: teóricamente, hasta 256) conforme se vayan necesitando. Veremos todo esto más adelante cuando estudiemos el sistema operativo *basado-en-el-objeto* del 432.

Facilidad de programación

Todavía hay otro problema creado por la proliferación de las computadoras y las aplicaciones de las computadoras, es la falta de gente capaz de programarlas. Esto es causa de los altos costes del desarrollo de software. Los métodos actuales son demasiado caros y la labor intensa. Se hace cada vez más importante el

utilizar software estándar y hardware modular en la construcción de tales sistemas, que permitan a los diseñadores de sistemas a cualquier nivel comprender el trabajo de los otros y ser mucho más productivos. Es decir, en vez de programar todas las piezas del sistema, se puede trabajar con piezas *preprogramadas*. El iAPX 432 incorpora un nuevo nivel de sofisticación en el cual un amplio conjunto de «primitivas» del sistema operativo forman parte realmente del conjunto de instrucciones hardware. En otras palabras, el 432 utiliza un sistema operativo basado-en silicio. Además, todas las operaciones código máquina son totalmente simétricas respecto a los operandos. Esto es, no hay casos especiales ni registros especiales con «personalidad» propia.

CONCEPTOS PRELIMINARES

Antes de entrar en detalles sobre el funcionamiento del iAPX 432 describiremos brevemente dos de los conceptos básicos que encierra su sistema operativo incorporado; a saber, las nociones de *objeto* y de *mensaje*. Los objetos son las «piezas» del sistema operativo, y los mensajes son la forma en la que se transmiten las órdenes y la información entre dichas piezas.

Objetos

Un *objeto* es una entidad que reside en memoria y que a la vez aloja y protege un conjunto de información relacionada (datos). Un *objeto* puede contener datos para el sistema operativo, o instrucciones del procesador, o datos de una aplicación particular. Cada objeto, en general, contiene un solo tipo de datos, aunque ciertos tipos de datos pueden ser muy complejos.

Los objetos en el sistema operativo viven su propia vida, actuando como trabajadores en una oficina, con sus títulos del trabajo, descripción del trabajo y tareas asignadas, Realizan estas tareas asignadas, y se comunican con los otros enviando y recibiendo «mensajes». Por ejemplo, varios objetos distintos que contengan vectores en coma flotante que representen partes de un proceso físico pueden intercambiar información sobre dicho proceso. ¡Los objetos tienen incluso su propio conjunto de números de teléfono privados en los que se les puede encontrar!

En realidad hay muchos tipos distintos de objetos en un sistema, cada uno especialista en la función a la que se dedica. Por ejemplo, cada *procesador* (hardware) del sistema tiene un único objeto (en memoria), llamado su *objeto procesador*. Este objeto actúa a modo de «agente» para el procesador, guardando datos vitales de las operaciones de dicho procesador (véase la fig. B.1). Para añadir un nuevo procesador al sistema basta con introducir el correspondiente objeto procesador. Por otro lado, cada trabajo del sistema se divide en subtrabajos llamados *procesos*, cada uno de los cuales está representado por un objeto (en memoria), llamado *objeto proceso*. El objeto proceso actúa a modo de «agente» para una porción particular de trabajo que necesite realizarse, almacenando parámetros de dicho trabajo. Otro tipo de objeto es el llamado *objeto distribuidor*. Actúa a modo de empleado de una oficina asignando los distintos objetos proceso (agentes para los trabajos) a objetos procesadores particulares

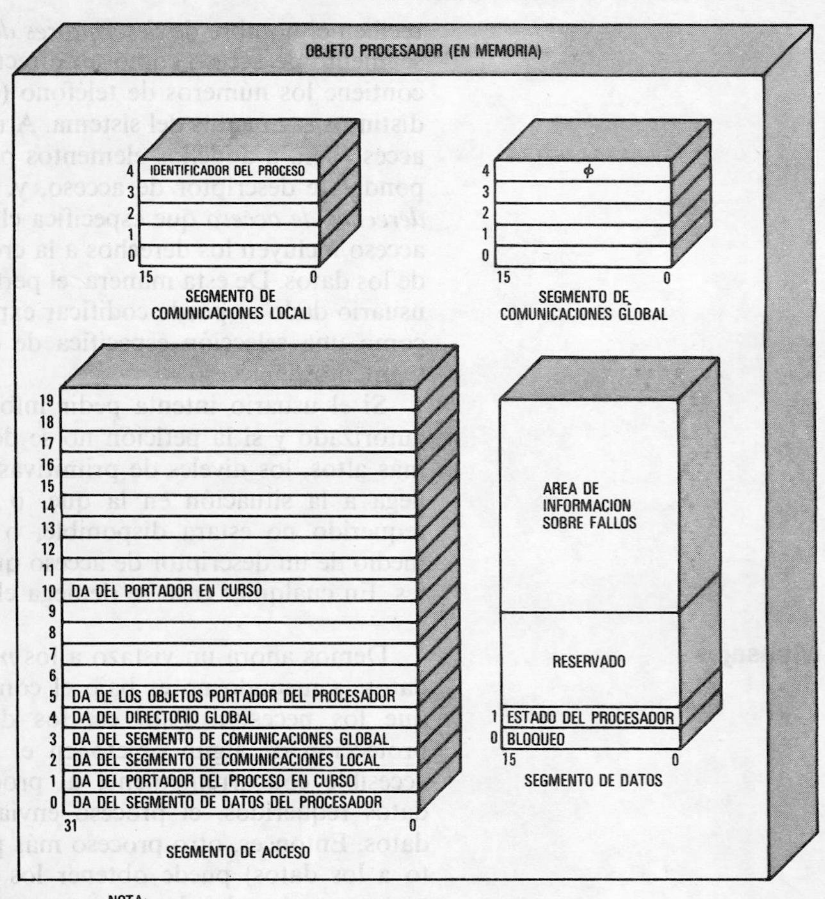


Figura B.1: Objetos procesador.

(agentes para la hardware, «trabajadores»). Existen otros tipos de objetos, como los *objetos de comunicación de ports, objetos portadores, objetos instrucción, objetos de ubicación de almacenamiento, objetos de tratamiento de errores y objetos de datos del usuario.*

La razón para tener todos estos objetos distintos es que cada objeto contiene un solo tipo de información. Con esto se consigue un control y protección excelentes de toda la información del sistema, porque se puede utilizar fácilmente una *verificación de tipo* dentro del esquema de protección. Es decir, cada acceso a datos implica una comprobación de que el dato es del tipo requerido. El esquema de protección de datos es muy interesante. Cada objeto consta de varios *segmentos,* algunos de los cuales reciben el nombre de *segmentos de acceso,* y los otros de *segmentos de datos*. Los segmentos de acceso permiten el acceso a la información y los segmentos de datos guardan los demás tipos de datos del sistema.

Un segmento de acceso es simplemente una lista o directorio de todo aquello a lo que puede acceder un objeto dado. Los elementos individuales almacenados en un segmento de acceso

reciben el nombre de *descriptores de acceso*. Puede imaginarse un segmento de acceso como un directorio privado de teléfonos que contiene los números de teléfono (descriptores de acceso) de los distintos segmentos del sistema. A un objeto dado se le permite el acceso sólo a aquellos elementos para los cuales tiene su correspondiente descriptor de acceso, y, de hecho, sólo para aquellos *derechos de acceso* que especifica el descriptor. Estos derechos de acceso incluyen los derechos a la creación, modificación o lectura de los datos. De esta manera, el perfil de derechos de acceso de un usuario dado se puede codificar explícitamente en el computador como una selección específica de descriptores de acceso en un segmento de acceso.

Si el usuario intenta pedir información para la que no está autorizado y si la petición no se deniega en los niveles software más altos, los niveles de primitivas lo detectan. En este caso, se llega a la situación en la que, o bien el descriptor de acceso requerido no estará disponible, o bien se intenta acceder por medio de un descriptor de acceso que no posee derechos suficientes. En cualquier caso se deniega el acceso.

Mensajes

Demos ahora un vistazo a los *mensajes*. Cuando se requieren datos, a menudo no es bajo el control de un proceso particular que los necesita. Esto es, los datos siempre están bajo la protección de algún objeto en el sistema que puede o no ser accesible concurrentemente al proceso dado. Para obtener los datos requeridos, el proceso envía un mensaje de petición de datos. Entonces, otro proceso más privilegiado (al menos respecto a los datos) puede obtener los datos, procesarlos tal vez, y empaquetarlos de alguna forma especial antes de enviarlos de vuelta al proceso original. De esta manera partes de la base de datos original que no se podrían ver se pueden mantener ocultos.

Los mensajes están formados en realidad por descriptores de acceso. Una petición de datos se representa por un descriptor de acceso apuntando a un segmento que describe lo que se necesita; y el mensaje de vuelta será un descriptor de acceso apuntando a un segmento que contenga información (quizás procesada).

Los mensajes como puede verse, proporcionan una manera de que varios procesos en un sistema trabajen juntos sin que entre en función más control que el estrictamente necesario. Los mensajes, además, proveen una buena forma de proteger los datosde accesos ilegales al mismo tiempo que permiten especificaciones detalladas de trabajos, haciendo que los datos «delicados» se comuniquen y se lleven fuera del sistema. En la práctica, enviar mensajes es algo semejante a llamar a subrutinas. Una diferencia importante es que en el caso de los mensajes, el proceso original no necesita esperar hasta que se haya recibido el mensaje para seguir con su trabajo, y, además, un proceso distinto puede a veces servir una petición enviada como mensaje.

LA FAMILIA iAPX 432

Veamos con más detalle cómo es y cómo trabaja el iAPX 432. El miembro principal de la familia iAPX 432 es un microprocesa-

dor de 32 bits llamado Procesador de Datos General, o GDP (General Data Processor). El GDP consta de dos chips, el Decodificador de Instrucción/Secuenciador de Microinstrucciones 43201, y la Unidad de Ejecución 43202. Mientras se escribía este libro, apareció otro miembro de la familia iAPX 432, el Procesador de Interfaz, ubicado en un chip cuyo número de modelo es el 43203. La figura B.2 muestra el diagrama de terminales de salida

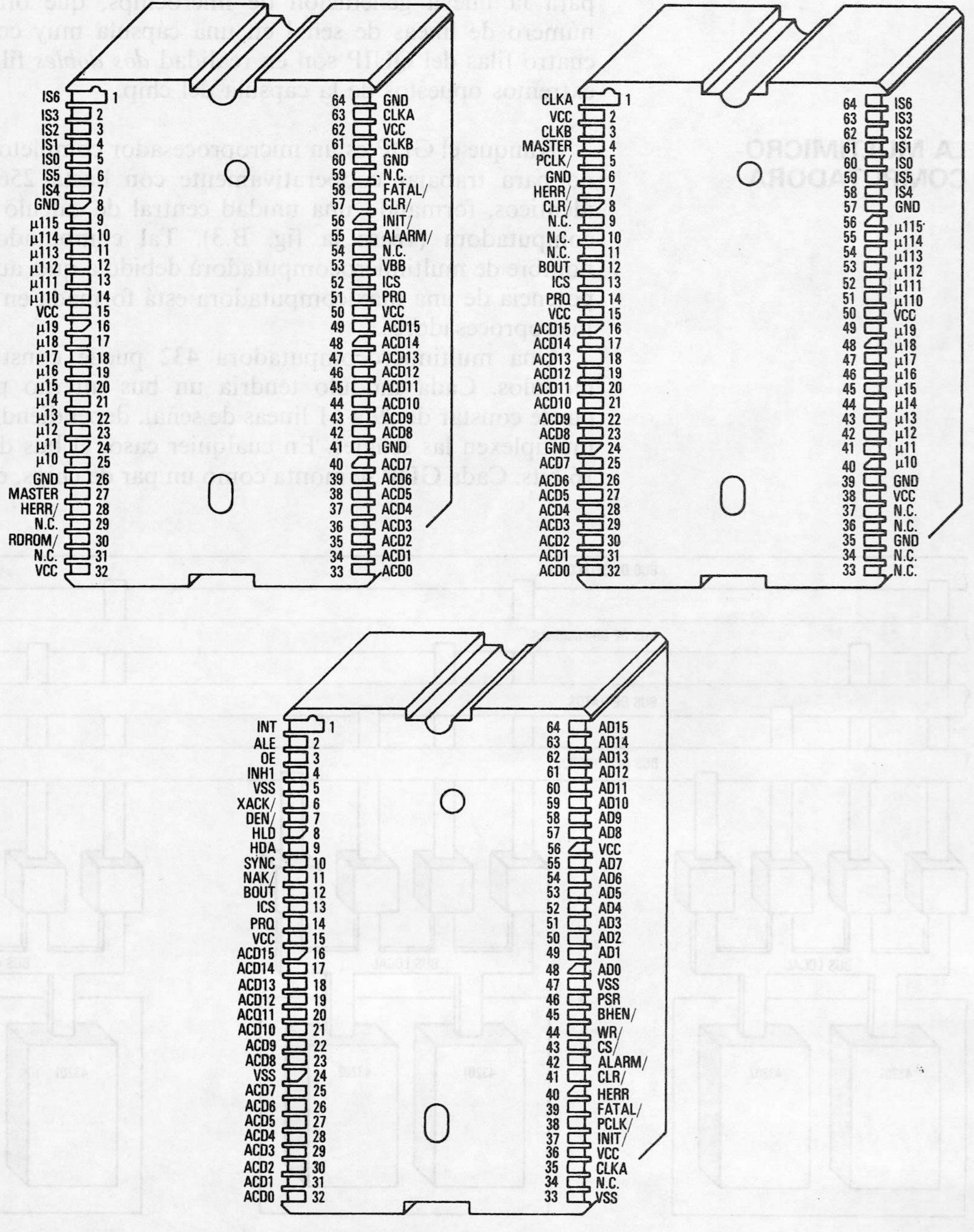


Figura B.2: Terminales de los chips de la familia iAPX.

de cada uno de estos chips. Cada uno de estos chips 432 se monta en cápsulas cuatro-en-línea (QUIP) de 64 terminales. *Cuatro-en-línea* quiere decir simplemente que los terminales se organizan en *cuatro* filas, contra la única fila de las cápsulas *una-en-línea* (SIP), o a las dos filas de las cápsulas *dos-en-línea* (DIP). La mayoría de los chips actuales se colocan en DIP, y hay cápsulas de resistencias que vienen en SIP.

La cápsula QUIP es un formató nuevo especialmente pensado para la nueva generación de microchips, que ofrece un gran número de líneas de señal en una cápsula muy compacta. Las cuatro filas del QUIP son en realidad *dos dobles* filas en los dos extremos opuestos de la cápsula del chip.

LA MULTIMICRO-COMPUTADORA

Aunque el GDP es un microprocesador completo, está diseñado para trabajar cooperativamente con hasta 256 chips GDP idénticos, formando una unidad central de cálculo de una gran computadora (véase la fig. B.3). Tal computadora recibe el nombre de multimicrocomputadora debido a que, aunque tiene la potencia de una maxicomputadora está formada en el fondo por microprocesadores.

Una multimicrocomputadora 432 puede constar de varios módulos. Cada módulo tendría un bus interno principal que puede constar de 29 a 61 líneas de señal, dependiendo de cómo se multiplexen las señales. En cualquier caso, el bus de datos tiene 16 bits. Cada GDP se monta como un par de chips, el Decodifica-

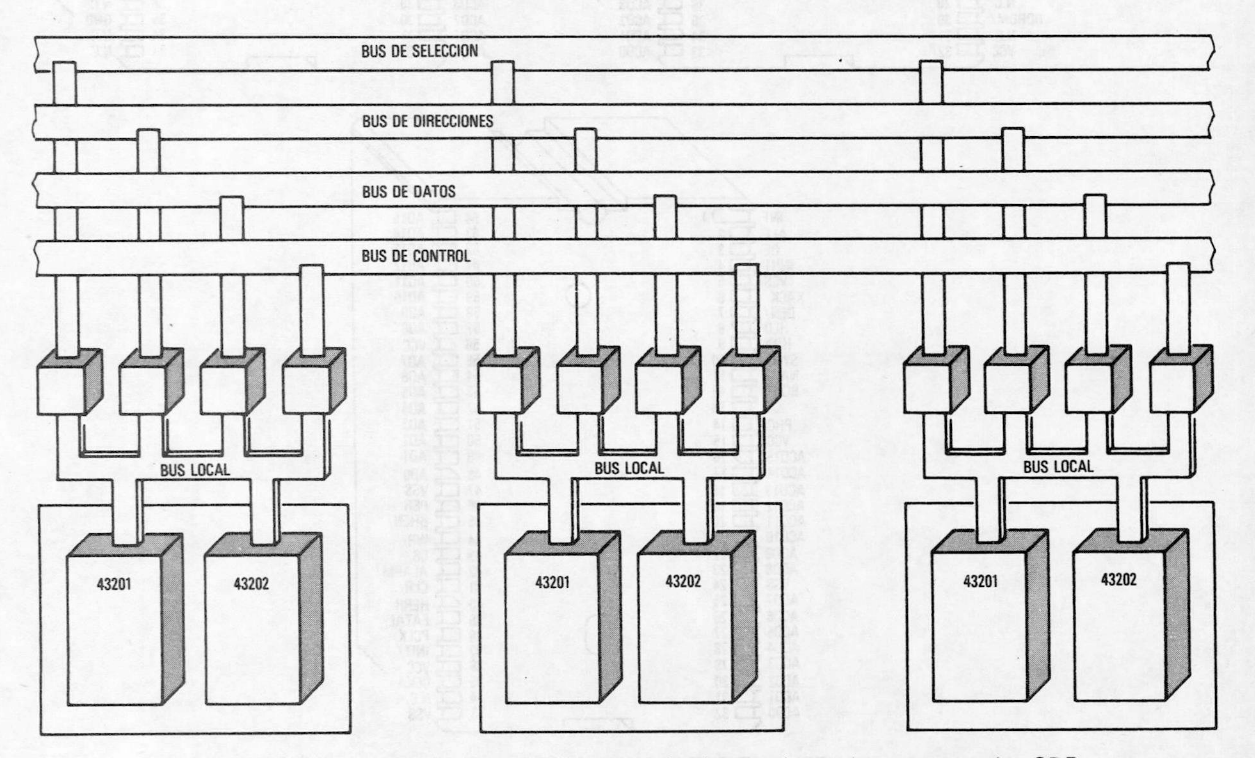


Figura B.3: Diagrama de bloques de un sistema con varios GDP.

dor de Instrucciones/Microsecuenciador (IDM) 43201, y la Unidad de Ejecución (EU) 43202. Ambos chips están uno junto al otro, con cerca de 23 líneas de señal entre ellos, pero no al bus principal. Desde el bus principal, algunas líneas van sólo al IDM, otras a la EU, y otras (por ejemplo, las de alimentación) a ambos. La figura B.4 muestra un diagrama de terminales de todo el GDP como una unidad.

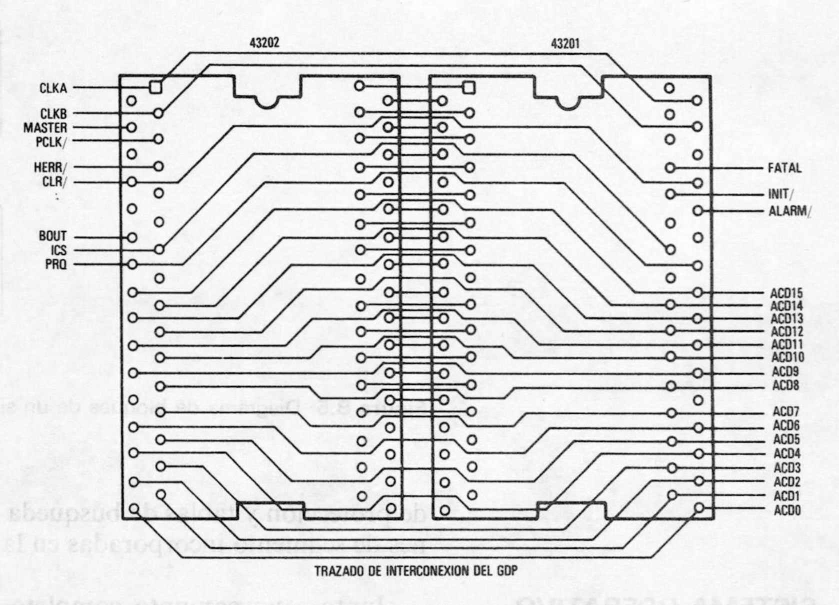


Figura B.4: Terminales de la unidad GDP.

Además de los GDP, la multimicrocomputadora 432 debería contener uno o más Procesadores de Interfaz. Un Procesador de Interfaz es al GDP lo que dos GDP son al sistema. Esto es, se conecta al bus principal del sistema y acepta el mismo tipo de órdenes interprocesador que un GDP, pero está conectado también a un segundo bus (el bus de E/S), al cual están conectados los procesadores de E/S y controladores. Los Procesadores de Interfaz proporcionan al sistema 432 una forma de comunicarse con el mundo exterior. Se podrían poner varios grupos de 8086, 8088 y 8089 sobre este bus de E/S o exterior (véase la figura B.5). Hay también «enganches» para integrar otros tipos de procesadores al bus principal conforme vayan saliendo al mercado. Hasta ahora, los planes sobre estos otros procesadores no se han hecho públicos (ni privadamente, al menos que nosotros sepamos).

Todos los procesadores del bus principal interno comparten la misma memoria, que puede ser muy grande. El 432 tiene un esquema de direccionamiento que le permite acceder de 2 a 40 octetos de direccionamiento de memoria, ¡lo que significa del orden de un trillón de octetos, o un millón de kilo-octetos! Se utiliza un esquema de segmentación con características especiales

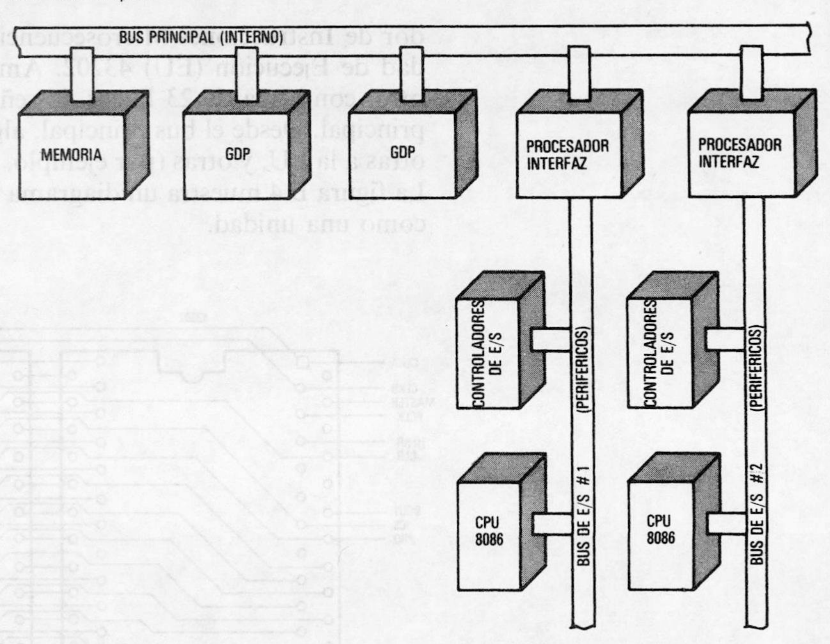


Figura B.5: Diagrama de bloques de un sistema con buses internos y de E/S.

de protección y tablas de búsqueda en memoria para las direcciones de segmento incorporadas en la hardware del procesador 432.

SISTEMA OPERATIVO BASADO-EN-OBJETOS Y BASADO-EN-SILICIO

Junto a un conjunto completo de instrucciones que realizan operaciones aritméticas y lógicas sobre un gran número de tipos de datos, el iAPX 432 tiene un juego de instrucciones ampliado para realizar funciones del sistema operativo. Recuérdese que el sistema operativo es el programa «madre» que supervisa las acciones de todos los programas de aplicación. El sistema operativo del 432 se basa en la noción de objeto, concepto que ya hemos visto anteriormente.

Las instrucciones del sistema operativo 432 trabajan examinando y modificando bits de elaboradas estructuras de datos en la memoria central de la computadora. Estas estructuras de datos reciben el nombre de objetos. Como ya se ha mencionado, estos objetos pueden estar formados por texto, números o incluso códigos de instrucción. Cada objeto está totalmente en la memoria del sistema computador 432 (aunque parte de esta memoria es un tipo de memoria rápida especial llamada «cache», *tampón*). En el 432 ¡no hay registros del procesador accesibles por el usuario! Todos los registros del procesador se guardan en el *segmento de datos* de tipos de objetos particulares llamados *objetos contexto* que ahora veremos. Los objetos-contexto tienen la interesante propiedad de que se crean y destruyen mientras el 432 va haciendo su trabajo. Así, ¡el 432 pierde frecuentemente todos los registros de los usuarios!

Es importante resaltar que aunque los objetos actúan inteli-

gentemente, ¡normalmente no contienen ni una pequeña mota de código de instrucción! De hecho sólo el *objeto instrucción* puede guardar instrucciones del procesador. Por el contrario la mayoría de los objetos contienen tablas de información que dicen al hardware lo que debe hacer a un nivel mucho más alto (casi como un lenguaje de alto nivel).

Para darnos una idea de como aparecen estos objetos en el interior, veamos con más detalle los objetos *contexto*.

Objetos contexto

Los objetos contexto juegan un papel clave en la ejecución del lenguaje máquina de 432. Están donde está la mayor parte de las acciones. Los *objetos contexto* están directamente asociados a cada procedimiento software (subrutina), pero sólo mientras el procedimiento particular se está ejecutando. Dicho de otra manera, se crea un contexto cuando se llama al procedimiento, y se destruye cuando se vuelve al sitio de llamada.

Internamente, los objetos contexto constan de los siguientes segmentos:

- Un segmento contexto de *acceso*
- Un segmento contexto de *datos*
- Un segmento contexto de *constantes*
- Un segmento contexto de *pila de operandos*

El segmento de acceso contiene un número variable de descriptores de acceso de 32 bits (como números telefónicos). Los 10 primeros son especiales y deben aparecer en el orden correcto, mientras que el resto depende de las necesidades del procedimiento particular que el contexto esté ejecutando.

El segmento contexto de datos contiene palabras de 16 bits que forman los registros de trabajo del procesador 432. Incluyen: una palabra de estado, un puntero de pila, un identificador de segmento de instrucción y un puntero de instrucciones. Tener todos los registros del usuario en memoria no es un concepto totalmente nuevo. El procesador de 16 bits de la Texas Instrument, por ejemplo, tiene un único registro de usuario en el procesador que apunta a un área de trabajo en memoria que contiene los otros registros del usuario. También es cierto que grandes máquinas como el Cyber CDC utilizan tablas en memoria para controlar lo que hacen.

El segmento de constantes contiene las constantes definidas en el procedimiento. Se intenta con esto que cualquier lenguaje en que se escriba el software pueda definir constantes por nombre (como en Pascal). El lenguaje que se propone es el Ada, un descendiente del Pascal. El Ada tiene una historia interesante. Se desarrolló como respuesta a una petición del Departamento de Defensa de los EE.UU. de un lenguaje universal para sistemas armamentísticos. Se ha convertido ahora en un lenguaje de uso todavía más universal aunque aún está en la etapa de afirmación de los estándares. Inicialmente se le dio el nombre de lenguaje Green (Verde), que más tarde se cambió por el de la primera

programadora de computadoras, lady Ada Augusta Byron, Countess Lovelace, hija del poeta Lord Byron. Al igual que el 432, el Ada es un lenguaje «fuertemente clasificado». Similarmente al 432, trata los trabajos en paquetes (objetos para el 432), y lleva incorporado la gestión de multitareas, aunque las multitareas de Ada y del 432 tienen poco en común.

El segmento de pila de operandos es opcional. Cuando está presente permite que el 432 opere como una máquina a pila (dentro de cada procedimiento). Las instrucciones aritméticas y lógicas permiten referenciar a, o bien la pila, o a los operandos almacenados en memoria como variables (o constantes). En el capítulo dedicado al NDP 8087 vimos que el 8087 utiliza una pila para guardar los operandos en coma flotante. Es fácil y eficaz escribir compiladores para máquinas de pila, por lo que dichas máquinas resultan muy útiles para lenguajes de alto nivel.

Las instrucciones del procesador 432 que operan sobre los objetos contexto son unas variaciones de CALL, RETURN o BRANCH. En las instrucciones CALL y RETURN, el objeto contexto en curso se destruye realmente, y se crea un nuevo objeto contexto, ¡una acción bastante drástica! La instrucción BRANCH funciona de una forma mucho más elegante, modificando ciertas cantidades guardadas en el objeto contexto.

Se pueden ver los objetos contexto como algo que describe el estado actual del procesador mientras está ejecutando un trabajo.

Hay más tipos de objetos en el 432, incluyendo objetos control de programa, así como objetos de tratamiento de errores, de datos y de instrucciones. Otros tipos de instrucciones del sistema operativo 432 se encargan de usar o destruir segmento de accesos y datos; de crear tipos de datos de alto nivel, y de examinar, transferir y modificar los descriptores de acceso. Donde se puede encontrar mejor información sobre este tema es en el manual Intel *iAPX 432 General Data Processor Architecture Reference Manual*. Esperamos que esta breve introducción le haya «abierto el apetito», e intente leerse el manual Intel.

CODIFICACION EN FLUJO DE BITS

El iAPX 432 codifica sus instrucciones en lo que se llama un *flujo de bits,* por contraposición al *flujo de octetos* utilizado por el iAPX 86,88. Significa que las instrucciones del iAPX 432 constan de cadenas de grupos de bits de cualquier longitud. Por ejemplo, hay una instrucción que se codifica con una cadena de ¡83 bits! del iAPX 436 para cada instrucción (ver tabla B.1). Los primeros tres campos se utilizan para especificar los operandos, y el último especifica el código de operación de la instrucción. Esto es parecido a la forma de entrar los datos primero y la operación después en las calculadoras HP. El último campo a veces ¡ni tan siquiera está presente en algunas instrucciones de 432 debido a que existe una sola instrucción cuyos operandos se ajusten a los descritos en los tres primeros campos!

Cada campo de bits tiene una longitud variable y se codifica de una forma muy ingeniosa, pero esto ya es otra historia.

El primer campo recibe el nombre de campo clase. Es

probablemente el más importante. Hablaremos un poco de este campo y dejaremos los otros (formato, referencia y código de operación) para otras lecturas más avanzadas que puede hacer el lector interesado. Los detalles de cómo trabajan estos cuatro campos son el secreto de los esquemas de protección de datos vitales del 432.

El campo clase dice cuántos operandos hay y de qué tamaño. Hay 41 *clases* diferentes, esto es, 41 posibles elecciones del número y tamaño de los operandos. Hay disponibles cinco longitudes diferentes de operandos para los datos y más longitudes para las bifurcaciones. Las posibles longitudes van de 8 a 80 bits. La tabla B.1 muestra las 41 clases distintas utilizadas por el 432.

Para ver cómo deben utilizarse estas clases, veamos algunos ejemplos:

1) La instrucción ADD para enteros de 16 bits del 432 tiene tres operandos, todos de 16 bits. Los dos primeros son las fuentes, y el tercero es el destino. Cuando se ejecuta la instrucción, se suman los dos operandos y el resultado se coloca en la posición especificada por el tercer operando.
2) La instrucción ADD para números enteros de 32 bits tiene también tres operandos, esta vez cada uno de 32 bits de tamaño.
3) La instrucción NOT EQUAL para números enteros de 32 bits tiene operandos de distintas longitudes. Tiene tres operandos, de los cuales los dos primeros son palabras (32 bits) y el último es un octeto (8 bits). La instrucción compara las dos palabras y devuelve en el octeto el resultado lógico de la operación.
4) La instrucción MOVE en formato real temporal (80 bits) tiene sólo dos operandos, ambos palabras ampliadas de 80 bits. El primero es la fuente y el segundo el destino.
5) La instrucción que pone a cero un carácter tiene un único operando (el destino) de 8 bits.
6) La instrucción RETURN no posee operandos.

CONCLUSIONES

Hemos examinado el iAPX 432 desde distintos puntos de vista: su papel en la sociedad, las estructuras de su sistema operativo y la codificación en sus instrucciones. Queda mucho que decir de todos estos temas y hay otros muchos temas del Intel 432 de los que no hemos hablado, pero esperamos que la discusión le haya dado una idea del experimento en arquitectura de computadoras que esto significa.

El 432 es probablemente el primero de una línea de desarrollos orientados a conseguir sistemas computadores complejos y de alta capacidad. El 432 está en el mercado a un precio todavía alto (unos 80.000 $ por sistema), pero se espera que baje hasta unos 1.000 $ por placa. No es todavía la respuesta última por varios factores. Por un lado, es difícil de inicializar. De hecho, para inicializar un sistema 432 se llena la memoria (desde un disco por ejemplo), para dejarla como si todos los 432 del sistema hubiesen estado trabajando durante un rato, y entonces se inicializan

ORDEN	LONGITUDES DE LOS OPERANDOS	CODIFICACION DE LA CLASE	OPERADORES DE LA CLASE
0	ninguna	001110	1
	bifurcar	101110	1
1	b, bifurcar	0000	2
	b	011110	3
	db	111110	9
	w	000001	8
	dw	100001	2
	ew	010001	2
2	b, b	110001	6
	b, db	001001	1
	db, b	101001	5
	db, db	1000	17
	db, w	011001	5
	db, ew	111001	2
	w, b	000101	7
	w, db	100101	5
	w, w	0100	13
	w, ew	010101	3
	dw, b	110101	3
	dw, dw	001101	3
	dw, ew	101101	1
	ew, b	011101	3
	ew, w	111101	3
	ew, dw	000011	1
	ew, ew	100011	4
3	b, b, b	010011	10
	db, db, b	1100	9
	db, db, db	0010	25
	db, db, w	110011	4
	db, w, w	001011	4
	w, w, b	1010	9
	w, w, w	0110	15
	w, w, ew	101011	4
	w, ew, ew	011011	4
	dw, dw, b	111011	3
	dw, dw, ew	000111	4
	dw, ew, ew	100111	4
	ew, w, ew	010111	4
	ew, dw, ew	110111	4
	ew, ew, b	001111	3
	ew, ew, ew	101111	5
	reservado	011111	
	reservado	111111	

NOTA:

b = octeto (8 bits)
db = doble octeto (16 bits)
w = palabra (32 bits)
dw = doble palabra (64 bits)
ew = palabra extendida (80 bits)
bifurcar = información sobre bifurcación (10 bits o 16 bits)

Tabla B.1: 41 clases de operandos distintas.

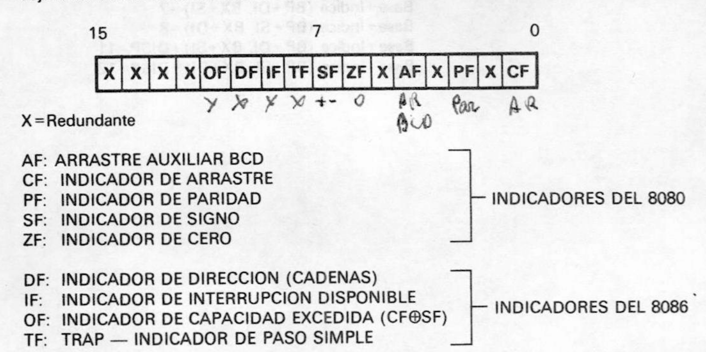
realmente los 432. Los 432 siempre están a mitad de algo, nunca al comienzo o al final. Por otro lado, un 432 no es demasiado rápido en cálculos corrientes. Comparado con un VAX resulta ser unas 10 veces más lento. Es pronto todavía para predecir cómo se comportará un 432 «adulto», cuando pueda utilizar toda su potencia porque tenga ya desarrollado el software necesario. Esto es un proceso lento. Así que ¡aguarde pacientemente a los nuevos desarrollos!

Apéndice C

Juego de instrucciones del 8086/8088



Las instrucciones que referencian al fichero registro de indicadores como un objeto de 16 bits usan el símbolo FLAGS para representar tal fichero:



RESUMEN DE OPERANDOS

Asignaciones de bits del campo «reg»

16 bits (w=1)	8 bits (w=0)	Segmento
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

RESUMEN DEL SEGUNDO OCTETO DE LA INSTRUCCION

mod	xxx	r/m

mod	Decalage
00	DISP=0* decalage-bajo y decalage-alto ausentes
01	DISP=decalage-bajo con extensión de signo a 16 bits, decalage-alto ausente
10	DISP=decalage-alto: decalage-bajo
11	r/m se trata como un campo «reg»

r/m	Dirección de los operandos
000	(BX)+(SI)+DISP
001	(BX)+(DI)+DISP
010	(BP)+(SI)+DISP
011	(BP)+(DI)+DISP
100	(SI)+DISP
101	(DI)+DISP
110	(BP)+DISP*
111	(BX)+DISP

DISP sigue al 2.º octeto de la instrucción (antes de los datos si se necesita).

* excepto si mod=00 y r/m=110; entonces EA=decalage-alto : decalage-bajo.

Ciclos de reloj de la dirección del operando (EA):
Sumar 4 ciclos para los operandos que ocupen palabras de DIRECCION IMPAR.
Desplazamiento inmediato=6.
Base+(BX, BP, SI, DI)=5
Base+DISP=9
Base+Indice (BP+DI, BX+SI)=7
Base+Indice (BP+SI, BX+DI)=8
Base+Indice (BP+DI, BX+SI)+DISP=11
Base+Indice (BP+SI, BX+DI)+DISP=12

TRANSFERENCIA DE DATOS

MOV: Mover

Registro/memoria a/de registro

1 0 0 0 1 0 d w	mod reg r/m

Ciclos de reloj: registro a registro 2
memoria a registro 8+EA
registro a memoria 9+EA

Inmediato a registro/memoria

1 1 0 0 0 1 1 w	mod 0 0 0 r/m	dato	dato si w=1

10+EA ciclos de reloj

Inmediato a registro

1 0 1 1 w reg	dato	dato si w=1

4 ciclos de reloj

Memoria a acumulador

1 0 1 0 0 0 0 w	dirección-baja	dirección-alta

10 ciclos de reloj

Acumulador a memoria

1 0 1 0 0 0 1 w	dirección-baja	dirección-alta

10 ciclos de reloj

Registro/memoria a registro de segmentación

1 0 0 0 1 1 1 0	mod 0 reg r/m

Ciclos de reloj: registro a registro 2
memoria a registro 8+EA

Registro de segmentación a registro/memoria

1 0 0 0 1 1 0 0	mod 0 reg r/m

Ciclos de reloj: registro a registro 2
registro a memoria 9+EA

PUSH: Introducir en pila

Registro/memoria

1 1 1 1 1 1 1 1	mod 1 1 0 r/m

Ciclos de reloj: registro 10
memoria 16+EA

Registro

0 1 0 1 0 reg

10 ciclos de reloj

Registro de segmentación

0 0 0 reg 1 1 0

10 ciclos de reloj

POP: Extraer de pila

Registro/memoria

1 0 0 0 1 1 1 1	mod 0 0 0 r/m

Ciclos de reloj: registro 8
memoria 17+EA

Registro

0 1 0 1 1 reg

8 ciclos de reloj

Registro de segmentación

0 0 0 reg 1 1 1

8 ciclos de reloj

XCGH: Intercambiar

Registro/memoria con registro

1 0 0 0 0 1 1 w	mod reg r/m

Ciclos de reloj: registro con registro 4
memoria con registro 17+EA

Registro con acumulador

1 0 0 1 0 reg

3 ciclos de reloj

IN: Entrar a AL/AX de

Port fijo

1 1 1 0 0 1 0 w	port

10 ciclos de reloj

Port variable (DX)

1 1 1 0 1 1 0 w

8 ciclos de reloj

OUT: Sacar de AL/AX a

Port fijo

1 1 1 0 0 1 1 w	port

10 ciclos de reloj

Port variable (DX)

1 1 1 0 1 1 1 w

8 ciclos de reloj

XLAT: Traducir octeto a AL

1 1 0 1 0 1 1 1

11 ciclos de reloj

LEA: Cargar EA a registro

1 0 0 0 1 1 0 1	mod reg r/m

2+EA ciclos de reloj

LDS: Cargar puntero a DS

1 1 0 0 0 1 0 1	mod reg r/m

16+EA ciclos de reloj

LES: Cargar puntero a ES

1 1 0 0 0 1 0 0	mod reg r/m

16+EA ciclos de reloj

LAHF: Cargar AH con los indicadores

1 0 0 1 1 1 1 1

4 ciclos de reloj

SAHF: Guardar AH en los indicadores

1 0 0 1 1 1 1 0

4 ciclos de reloj

PUSHF: PUSH indicadores

1 0 0 1 1 1 0 0

10 ciclos de reloj

POPF: POP indicadores

1 0 0 1 1 1 0 1

8 ciclos de reloj

ARITMETICAS

ADD: Suma

Reg./memoria con registro a cualquiera de ellos

0 0 0 0 0 0 d w	mod reg r/m	

Ciclos de reloj:	registro a registro	3
	memoria a registro	9+EA
	registro a memoria	16+EA

Inmediato a registro/memoria

1 0 0 0 0 0 s w	mod 0 0 0 r/m	dato	dato si s:w=01

Ciclos de reloj:	inmediato a registro	4
	inmediato a memoria	17+EA

Inmediato de registro/memoria

1 0 0 0 0 0 s w	mod 1 0 1 r/m	dato	dato si s:w=01

Ciclos de reloj:	inmediato de registro	4
	inmediato de memoria	17+EA

Inmediato de acumulador

0 0 1 0 1 1 0 w	dato	dato si w=1

4 ciclos de reloj

SBB: Resta con adeudo

Reg./memoria y registro a cualquiera de ellos

0 0 0 1 1 0 d w	mod reg r/m

Ciclos de reloj:	registro de registro	3
	memoria de registro	9+EA
	registro de memoria	16+EA

Inmediato de registro/memoria

1 0 0 0 0 0 s w	mod 0 1 1 r/m	dato	dato si w=01

Ciclos de reloj:	inmediato de registro	4
	inmediato de memoria	17+EA

Inmediato de acumulador

0 0 0 1 1 1 0 w	dato	dato si w=1

4 ciclos de reloj

DEC: Decrementar

Registro/memoria

1 1 1 1 1 1 1 w	mod 0 0 1 r/m

Ciclos de reloj:	registro	2
	memoria	15+EA

Registro

0 1 0 0 1 reg

2 ciclos de reloj

NEG: Cambio de signo

1 1 1 1 0 1 1 w	mod 0 1 1 r/m

Ciclos de reloj:	registro	3
	memoria	16+EA

CMP: Comparación

Registro/memoria y registro

0 0 1 1 1 0 d w	mod reg r/m

Ciclos de reloj:	registro con registro	3
	memoria con registro	9+EA
	registro con memoria	9+EA

Inmediato a acumulador

0 0 0 0 0 1 0 w	dato	dato si w=1

4 ciclos de reloj

(Continúa en la página siguiente)

ADC: Suma con acarreo
Reg./memoria con registro a cualquiera de ellos

0 0 0 1 0 0 d w	mod reg r/m

Ciclos de reloj:	registro a registro	3
	memoria a registro	9+EA
	registro a memoria	16+EA

Inmediato a registro/memoria

1 0 0 0 0 0 s w	mod 0 1 0 r/m	dato	dato si s:w=01

Ciclos de reloj:	inmediato a registro	4
	inmediato a memoria	17+EA

Inmediato a acumulador

0 0 0 1 0 1 0 w	dato	dato si w=1

4 ciclos de reloj

INC: Incrementar
Registro/memoria

1 1 1 1 1 1 1 w	mod 0 0 0 r/m

Ciclos de reloj:	registro	2
	memoria	15+EA

Registro

0 1 0 0 0 reg

2 ciclos de reloj

AAA: Ajuste ASCII para la suma

0 0 1 1 0 1 1 1

4 ciclos de reloj

DAA: Ajuste decimal para la suma

0 0 1 0 0 1 1 1

4 ciclos de reloj

SUB: Resta
Reg./memoria y registro a cualquiera de ellos

0 0 1 0 1 0 d w	mod reg r/m

Ciclos de reloj:	registro de registro	3
	memoria de registro	9+EA
	registro de memoria	16+EA

Inmediato con registro/memoria

1 0 0 0 0 0 s w	mod 1 1 1 r/m	dato	dato si s:w=01

Ciclos de reloj:	inmediato con registro	4
	inmediato con memoria	17+EA

Inmediato con acumulador

0 0 1 1 1 1 0 w	dato	dato si w=1

4 ciclos de reloj

AAS: Ajuste ASCII para la resta

0 0 1 1 1 1 1 1

4 ciclos de reloj

DAS: Ajuste decimal para la resta

0 0 1 0 1 1 1 1

4 ciclos de reloj

MUL: Multiplicación (sin signo)

1 1 1 1 0 1 1 w	mod 1 0 0 r/m

Ciclos de reloj:	8 bits	71+EA
	16 bits	124+EA

IMUL: Multiplicación entera (con signo)

1 1 1 1 0 1 1 w	mod 1 0 1 r/m

Ciclos de reloj:	8 bits	90+EA
	16 bits	144+EA

AAM: Ajuste ASCII para multiplicar

1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0

83 ciclos de reloj

DIV: División (sin signo)

1 1 1 1 0 1 1 w	mod 1 1 0 r/m

Ciclos de reloj:	8 bits	90+EA
	16 bits	155+EA

IDIV: División entera (con signo)

1 1 1 1 0 1 1 w	mod 1 1 1 r/m

Ciclos de reloj:	8 bits	112+EA
	16 bits	177+EA

ADD: Ajuste ASCII para dividir

1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0

60 ciclos de reloj

CBW: Conversión de byte a palabra

1 0 0 1 1 0 0 0

2 ciclos de reloj

CWD: Conversión de palabra a doble palabra

1 0 0 1 1 0 0 1

5 ciclos de reloj

LOGICAS

NOT: Negación

1 1 1 1 0 1 1 w	mod 0 1 0 r/m

Ciclos de reloj:	registro	3
	memoria	16+EA

SHL/SAL: Desplazamiento lógico/aritmético a la izquierda

1 1 0 1 0 0 v w	mod 1 0 0 r/m

Ciclos de reloj:	registro bit-simple	2
	memoria bit-simple	15+EA
	registro bit-variable	8+4/bit
	memoria bit-variable	20+EA+4/bit

SHR: Desplazamiento lógico a la derecha

1 1 0 1 0 0 v w	mod 1 0 1 r/m

Ciclos de reloj:	registro bit-simple	2
	memoria bit-simple	15+EA
	registro bit-variable	8+4/bit
	memoria bit-variable	20+EA+4/bit

SAR: Desplazamiento aritmético a la derecha

1 1 0 1 0 0 v w	mod 1 1 1 r/m

Ciclos de reloj:	registro bit-simple	2
	memoria bit-simple	15+EA
	registro bit-variable	8+4/bit
	memoria bit-variable	20+EA+4/bit

(Continúa en la página siguiente)

ROL: Rotar a la izquierda

1 1 0 1 0 0 v w	mod 0 0 0 r/m

Ciclos de reloj:
- registro bit-simple 2
- memoria bit-simple 15+EA
- registro bit-variable 8+4/bit
- memoria bit-variable 20+EA+4/bit

ROR: Rotar a la derecha

1 1 0 1 0 0 v w	mod 0 0 1 r/m

Ciclos de reloj:
- registro bit-simple 2
- memoria bit-simple 15+EA
- registro bit-variable 8+4/bit
- memoria bit-variable 20+EA+4/bit

RCL: Rotar a la izquierda con acarreo

1 1 0 1 0 0 v w	mod 0 1 0 r/m

Ciclos de reloj:
- registro bit-simple 2
- memoria bit-simple 15+EA
- registro bit-variable 8+4/bit
- memoria bit-variable 20+EA+4/bit

RCR: Rotar a la derecha con acarreo

1 1 0 1 0 0 v w	mod 0 1 1 r/m

Ciclos de reloj:
- registro bit-simple 2
- memoria bit-simple 15+EA
- registro bit-variable 8+4/bit
- memoria bit-variable 20+EA+4/bit

AND: Producto lógico
Reg./memoria y registro a cualquiera de ellos

0 0 1 0 0 0 d w	mod reg r/m

Ciclos de reloj:
- registro a registro 3
- memoria a registro 9+EA
- registro a memoria 16+EA

Inmediato a registro/memoria

1 0 0 0 0 0 0 w	mod 1 0 0 r/m	dato	dato si w=1

Ciclos de reloj:
- inmediato a registro 4
- inmediato a memoria 17+EA

Inmediato a acumulador

0 0 1 0 0 1 0 w	dato	dato si w=1

4 ciclos de reloj

TEST: AND función a indicadores, sin resultado
Registro/memoria y registro

1 0 0 0 0 1 0 w	mod reg r/m

Ciclos de reloj:
- registro a registro 3
- registro con memoria 9+EA

Dato inmediato y registro/memoria

1 1 1 1 0 1 1 w	mod 0 0 0 r/m	dato	dato si w=1

Ciclos de reloj:
- inmediato con registro 4
- inmediato con memoria 10+EA

Dato inmediato y acumulador

1 0 1 0 1 0 0 w	dato	dato si w=1

4 ciclos de reloj

OR: Suma lógica
Registro/memoria o registro a cualquiera de ellos

0 0 0 0 1 0 d w	mod reg r/m

Ciclos de reloj:
- registro a registro 3
- memoria a registro 9+EA
- registro a memoria 16+EA

Inmediato a registro/memoria

1 0 0 0 0 0 0 w	mod 0 0 1 r/m	dato	dato si w=1

Ciclos de reloj:
- inmediato a registro 4
- inmediato a memoria 17+EA

Inmediato a acumulador

0 0 0 0 1 1 0 w	dato	dato si w=1

4 ciclos de reloj

XOR: O exclusiva
Reg./memoria y registro a cualquiera de ellos

0 0 1 1 0 0 d w	mod reg r/m

Ciclos de reloj:
- registro a registro 3
- memoria a registro 9+EA
- registro a memoria 16+EA

Inmediato a registro/memoria

1 0 0 0 0 0 0 w	mod 1 1 0 r/m	dato	dato si w=1

Ciclos de reloj:
- inmediato a registro 4
- inmediato a memoria 17+EA

Inmediato a acumulador

0 0 1 1 0 1 0 w	dato	dato si w=1

4 ciclos de reloj

TRATAMIENTO DE CADENAS

REP: Repetir

1 1 1 1 0 0 1 z

6 ciclos de reloj/bucle

MOVS: Mover cadena

1 0 1 0 0 1 0 w

17 ciclos de reloj

CPMS: Comparar cadena

1 0 1 0 0 1 1 w

22 ciclos de reloj

SCAS: Buscar en cadena

1 0 1 0 1 1 1 w

15 ciclos de reloj

LODS: Cargar cadena

1 0 1 0 1 1 0 w

12 ciclos de reloj

STOS: Guardar cadena

1 0 1 0 1 0 1 w

10 ciclos de reloj

CONTROL DE PROGRAMA

NOTA: La reinicialización de la cola no está incluida en la información de los ciclos de reloj de las operaciones de control de programa. Para contar la carga de instrucciones, sumar 8 ciclos de reloj.

CALL: Llamada

Directa en el segmento

1 1 1 0 1 0 0 0	decalage-bajo	decalage-alto

11 ciclos de reloj

Indirecta en el segmento

1 1 1 1 1 1 1 1	mod 0 1 0 r/m

13 + EA ciclos de reloj

Directa intersegmento

1 0 0 1 1 0 1 0	desplazamiento-bajo	desplazamiento-alto
	segmento-bajo	segmento-alto

20 ciclos de reloj

Indirecta intersegmento

1 1 1 1 1 1 1 1	mod 0 1 1 r/m

29 + EA ciclos de reloj

JMP: Salto incondicional

Directo en segmento

1 1 1 0 1 0 0 1	decalage-alto	decalage-bajo

7 ciclos de reloj

Directo en segmento-corto

1 1 1 0 1 0 1 1	decalage

7 ciclos de reloj

Indirecto en segmento

1 1 1 1 1 1 1 1	mod 1 0 0 r/m

7 + EA ciclos de reloj

Directo intersegmento

1 1 1 0 1 0 1 0	desplazamiento-bajo	desplazamiento-alto
	segmento-bajo	segmento-alto

7 ciclos de reloj

Indirecto intersegmento

1 1 1 1 1 1 1 1	mod 1 0 1 r/m

16 + EA ciclos de reloj

RET: Vuelta de una CALL

En el segmento

1 1 0 0 0 0 1 1

8 ciclos de reloj

En el segmento añadiendo un inmediato a SP

1 1 0 0 0 0 1 0	datos-bajo	datos-alto

12 ciclos de reloj

Intersegmento

1 1 0 0 1 0 1 1

18 ciclos de reloj

Intersegmento, añadiendo un inmediato a SP

1 1 0 0 1 0 1 0	datos-bajo	datos-alto

17 ciclos de reloj

JE/JZ: Saltar si igual/cero

0 1 1 1 0 1 0 0	decalage

Ciclos de reloj: Si hay salto 8
Si no hay salto 4

JL/JNGE: Saltar si menor/no mayor o igual

0 1 1 1 1 1 0 0	decalage

Ciclos de reloj: Si hay salto 8
Si no hay salto 4

JLE/JNG: Saltar si menor o igual/no mayor

0 1 1 1 1 1 1 0	decalage

Ciclos de reloj: Si hay salto 8
Si no hay salto 4

JB/JNAE: Saltar si por debajo/no por encima o igual

0 1 1 1 0 0 1 0	decalage

Ciclos de reloj: Si hay salto 8
Si no hay salto 4

JBE/JNA: Saltar si por debajo o igual/no por encima

0 1 1 1 0 1 1 0	decalage

Ciclos de reloj: Si hay salto 8
Si no hay salto 4

JP/JPE: Saltar si paridad/paridad par

0 1 1 1 1 0 1 0	decalage

Ciclos de reloj: Si hay salto 8
Si no hay salto 4

JO: Saltar si capacidad excedida

0 1 1 1 0 0 0 0	decalage

Ciclos de reloj: Si hay salto 8
Si no hay salto 4

JS: Saltar según signo

0 1 1 1 1 0 0 0	decalage

Ciclos de reloj: Si hay salto 8
Si no hay salto 4

(Continúa en la página siguiente)

JNE/JNZ: Salto si no igual/no cero

0 1 1 1 0 1 0 1	decalage

Ciclos de reloj: Se salta 8
No se salta 4

JNL/JGE: Salto si no menor/mayor o igual

0 1 1 1 1 1 0 1	decalage

Ciclos de reloj: Se salta 8
No se salta 4

JNLE/JG: Salto si no menor o igual/mayor

0 1 1 1 1 1 1 1	decalage

Ciclos de reloj: Se salta 8
No se salta 4

JNB/JAE: Salto si no por abajo/encima o igual

0 1 1 1 0 0 1 1	decalage

Ciclos de reloj: Se salta 8
No se salta 4

JNBE/JA: Salto si no por debajo o igual/por encima

0 1 1 1 0 1 1 1	decalage

Ciclos de reloj: Se salta 8
No se salta 4

JNP/JPO: Salto si no paridad/paridad impar

0 1 1 1 1 0 1 1	decalage

Ciclos de reloj: Se salta 8
No se salta 4

JNO: Salto si no hay capacidad excedida

0 1 1 1 0 0 0 1	decalage

Ciclos de reloj: Se salta 8
No se salta 4

JNS: Salto si no signo

0 1 1 1 1 0 0 1	decalage

Ciclos de reloj: Se salta 8
No se salta 4

LOOP: Buclar CX veces

1 1 1 0 0 0 1 0	decalage

Ciclos de reloj: Se salta 9
No se salta 5

LOOPZ/LOOPE: Buclar mientras cero/igual

1 1 1 0 0 0 0 1	decalage

Ciclos de reloj: Se salta 11
No se salta 5

LOOPNZ/LOOPNE: Buclar mientras distinto de cero/no igual

1 1 1 0 0 0 0 0	decalage

Ciclos de reloj: Se salta 11
No se salta 5

JCXZ: Salto si CX cero

1 1 1 0 0 0 1 1	decalage

Ciclos de reloj: Se salta 9
No se salta 5

OPERACIONES DE TRANSFERENCIA CONDICIONALES EN EL 8086

Instrucción	Condición	Interpretación
JE or JZ	ZF=1	«igual» o «cero»
JL or JNGE	(SF xor OF) −1	«menor» o «no mayor ni igual»
JLE or JNG	((SP xor OF) or ZF) =1	«menor o igual» o «no mayor»
JB or JNAE	CF=1	«abajo» o «no arriba ni igual»
JBE or JNA	(CF or ZP) =1	«abajo o igual» o «no arriba»
JP or JPE	PF=1	«paridad» o «paridad par»
J0	OF=1	«capacidad excedida»
JS	SF=1	«signo»
JNE or JNZ	ZF=0	«no igual» o «no cero»
JNL or JGE	(SF xor OF) =0	«no menor» o «mayor o igual»
JNLE or JG	((SF xor OF) or ZF) =0	«no menor ni igual» o «mayor»
JNB or JAE	CF=0	«no abajo» o «arriba o igual»
JNBE or JA	(CF or ZF) =0	«no abajo igual» o «arriba»
JNP or JPO	PF=0	«sin paridad» o «paridad impar»
JNO	OF=0	«capacidad no excedida»
JNS	SF=0	«sin signo»

«Arriba» y «abajo» se refieren a la relación entre dos valores sin signo y mientras que «mayor» y «menor» se refieren a la relación entre dos valores con signo.

INT: Interrupción
Tipo especificado

1 1 0 0 1 1 0 1	tipo

50 ciclos de reloj

Tipo 3

1 1 0 0 1 1 0 0

51 ciclos de reloj

INTO: Interrupción si se excede de capacidad

1 1 0 0 1 1 1 0

52 ciclos de reloj si la hay 4 ciclos si no la hay

IRET: Retorno de interrupción

1 1 0 0 1 1 1 1

24 ciclos de reloj

CONTROL DEL PROCESADOR

CLC: Borrar adeudo

1 1 1 1 1 0 0 0

2 ciclos de reloj

STC: Poner a 1 adeudo

1 1 1 1 1 0 0 1

2 ciclos de reloj

CMC: Complementar adeudo

1 1 1 1 0 1 0 1

2 ciclos de reloj

NOP: No operación

1 0 0 1 0 0 0 0

3 ciclos de reloj

CLD: Borrar dirección

1 1 1 1 1 1 0 0

2 ciclos de reloj

STD: Establecer dirección

1 1 1-1 1 1 0 1

2 ciclos de reloj

CLI: Borrar interrupción

1 1 1 1 1 0 1 0

2 ciclos de reloj

STI: Activar interrupción

1 1 1 1 1 0 1 1

2 ciclos de reloj

HLT: Parar

1 1 1 1 0 1 0 0

2 ciclos de reloj

WAIT: Espera

1 0 0 1 1 0 1 1

3 ciclos de reloj

LOCK: Prefijo de bloqueo del bus

1 1 1 1 0 0 0 0

2 ciclos de reloj

ESC: Escape (a un dispositivo externo)

1 1 0 1 1 x x x	mod x x x r/m

7+EA ciclos de reloj

Notas:
si d=1, «a»; si d=0, «de»
si w=1, instrucción de 1 palabra; si w=0, instrucción de 1 octeto
si s:w=01, operando formado por dato inmediato de 16 bits
si s:w=11, operando de 16 bits formado por un dato inmediato de 1 octeto con extensión de signo
si v=0, «contado»=1; si v=1, «contado» en (CL)
x=redundante
z se usa para algunas primitivas de cadenas para comparación con ZF

AL=acumulador de 8 bits
AX=acumulador de 16 bits
CX=registro contador
DS=segmento de datos
DX=registro de port variable
ES=segmento extra
Por encima/por debajo, se refiere a valores absolutos
Mayor=más positivo
Menor=menos positivo (más negativo). Valores con signo
Véase página 1, Resumen de operandos
Véase página 2, Resumen anulación segmentos

Indice

OTRAS OBRAS DE INTERES PUBLICADAS POR OSBORNE/McGRAW-HILL

ADAM OSBORNE: Guía del comprador de sistemas de gestión
ADAM OSBORNE: Guía del ordenador personal PET/CBM
ANNIE FOX: BASIC básico. Guía para principiantes
CASTLEWITZ: Introducción al Visicalc
JOHN HEILBORN: Programas para ciencias e ingeniería. Edición APPLE II
JOHN HEILBORN: VIC 20. Guía del usuario
LON POOLE: Algunos programas de uso común en BASIC
LON POOLE: Algunos programas de uso común en BASIC. Edición APPLE II
LON POOLE: Algunos programas de uso común en BASIC. Edición ATARI
LON POOLE: Algunos programas de uso común en BASIC. Edición IBM
LON POOLE: Algunos programas de uso común en BASIC. Edición PET/CBM
LON POOLE: Algunos programas de uso común en BASIC. Edición TRS-80
LON POOLE: Algunos programas de uso común en PASCAL
LON POOLE: APPLE II. Guía del ususario
LON POOLE: Programas prácticos en BASIC
LON POOLE: Programas prácticos en BASIC. Edición APPLE II
LON POOLE: Programas prácticos en BASIC. Edición IBM
LON POOLE: Programas prácticos en BASIC. Edición TRS-80
LON POOLE: Programas prácticos en PASCAL
LYLE J. GRAHAM: IBM/PC. Guía del usuario
ROBERT MOTTOLA: Programación en lenguaje ensamblador para el APPLE II
TOM HOGAN: Sistema operativo CP/M. Guía del usuario (2.ª ed.)
WALTER ETTLIN: Introducción al WORDSTAR
MITCHELL WAITE: Introducción al procesamiento de palabras

DISCOGUIAS PUBLICADOS POR OSBORNE/McGRAW-HILL

CURTIS A. INGRAHAM: Discoguía para CP/M
DAVID A. WILSON: Discoguía para IBM/PC
DAVID A. WILSON: Discoguía para VISICALC
JOHN TAYLOR: Discoguía para ATARI 400/800
ZELDA GIFFORD: Discoguía para APPLE II

OTRAS OBRAS DE INTERES PUBLICADAS POR BYTE-BOOKS/McGRAW-HILL

ABELSON: APPLE LOGO
BOWLES: Introducción al UCSD Pascal
CIARCIA: Construya una computadora basado en el Z-80 (Guía de diseño y funcionamiento)
KAMINS: Usted y la microcomputadora (Una introducción humanizada a la microinformática)
LEWART: Programas de ciencias e ingeniería para microcomputadoras SINCLAIR ZX81 compatibles con el ZX SPECTRUM
PECKHAM: BASIC para APPLE II. Manual práctico
SIKONOWIZ: Introducción al IBM/PC

ISBN: 968-45/-628-2



BYTE BOOKS

Introducción al MICROPROCESADOR **8086/8088** (16 Bit) Morgan—Waite